

eVM™ for Windows User Manual

Copyright © 2010 TenAsys Corporation All Rights Reserved

No part of this manual may be copied, duplicated, reprinted, and stored in a retrieval system by any means, mechanical or electronic, without the written permission of the copyright owner.

First Printing: February 2010

Published in the United States by

TenAsys Corporation

1400 NW Compton Drive

Suite 301

Beaverton, OR 97006, USA

Attempts have been made to properly reference all copyrighted, registered, and trademarked material. All copyrighted, registered, and trademarked material remains the property of the respective owners.

The publisher, author, and reviewers make no warranty for the correctness or for the use of this information, and assume no liability for direct or indirect damages of any kind arising from the information contained herewith, technical interpretation or technical explanations, for typographical or printing errors, or for any subsequent changes in this article.

The publisher and author reserve the right to make changes in this publication without notice and without incurring any liability.

Windows® and Windows Vista® are registered trademarks of Microsoft® Corporation.

TenAsys®, INtime®, and iRMX® are registered trademarks of TenAsys® Corporation.

eVM™ is a trademark of TenAsys® Corporation.

Intel® is a registered trademark of Intel Corporation.

All other company names and products herein may be trademarks of their respective owners.

Table of Contents

1	Introduction	7
1.1	Overview of eVM for Windows.....	7
2	Quick Start Guide	9
3	Product Technical Description	10
3.1	What is Real-Time embedded virtualization?.....	10
3.2	Theory of Operation of eVM.....	10
3.2.1	What is eVM.....	10
3.2.2	Introduction to virtualization.....	10
3.2.3	eVM and Virtualization technology	11
3.2.4	eVM system requirements (brief overview)	11
3.2.5	Performance: How eVM addresses RTOS / Windows Application Needs	12
3.2.6	Communications between Windows and the Guest: Virtual Device Interfaces.....	12
3.2.7	eVM Architecture: Partitioning the PC between Windows and Guest OS	13
3.2.8	Resources that are not Isolated between Windows and the Guest OS.....	14
3.2.9	Device Management	14
3.2.10	What Happens when a Device is assigned to eVM.....	19
4	Installation	21
4.1	Installation of eVM for Windows.....	21
4.2	Virtual Ports	39
4.2.1	Installation of Virtual Serial Ports	40
4.2.2	Configuring the Virtual Ethernet Device	47
4.3	Running the Sample Guest RTOS, iRMX.....	55
4.4	Installation of a Guest OS.....	59
4.4.1	Configure your existing system.....	60
4.4.2	Make a disk image of your system.....	60
4.4.3	Make a guest configuration	60
4.4.4	Installing your operating system from installation materials	61
4.5	Specific steps for certain Guest OSes.....	61
4.5.1	Installation and configuration of QNX 6.4	61
4.5.2	Installation and configuration of Windows CE.....	65

4.6	Driver configuration and setup	65
4.6.1	eVM Device Manager	65
4.6.2	Passing a Legacy IRQ Device to eVM	70
4.6.3	Conflict Example when Passing Legacy IRQ Device to eVM	78
4.6.4	Passing an MSI Device to eVM	80
4.6.5	Bus Master Warning Example	83
4.6.6	Passing a Device back to Windows	85
A.	Production installation	90
A.1	Saving the configuration with the /e: option	90
A.2	Restoring or importing a known configuration	91
A.3	Assigning devices to eVM on the command line	93
	PCI devices	93
A.4	Command Line Installation of Virtual Serial Ports	93
A.5	Debug logging	93
B.	eVM Virtual Machine	95
B.1	System Requirements	95
B.2	Base Virtual Machine Specification	95
	CPU	95
	Memory	95
	Interrupt controller (8259A PIC)	96
	System Timer (8254 PIT)	96
	CMOS & Real Time Clock	96
	Port 92	96
	Serial ports (16550 UART)	97
	DMA controller (8237 DMA controller)	97
	Floppy Disk Controller	97
	IDE controller	97
	PCI Host Bridge (440FX Host Bridge, PCI device 0)	97
	ISA Bridge (440FX ISA Bridge, PCI ID 8086/7000)	97
	EIDE Controller (PCI ID 8086/7010)	97
	ACPI Controller (PIIX4 ACPI Controller, PCI ID 8086/7113)	98
B.3	Optional Virtual Machine Components	98

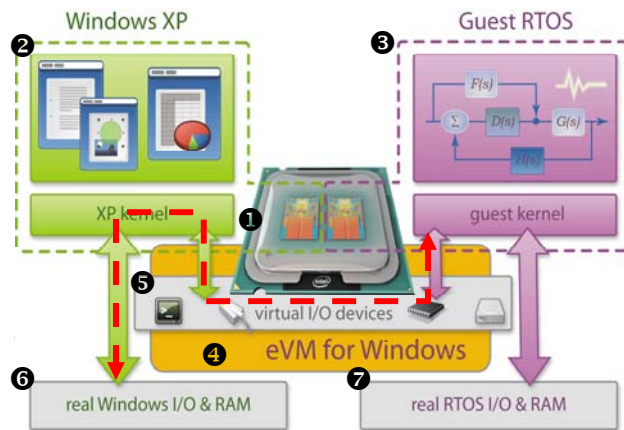
Ethernet controller	98
Shared memory device	98
B.4 BIOS	98
C. Guest OSes Currently Enabled:	99
D. VtProbe	100
E. Modifying device drivers for bus-master devices	102
E.1 Discovering the guest physical base address	102
E.2 Using the guest physical base offset value	102
F. Windows XP Installation of eVM for Windows	103
G. Windows XP Configuration of the Virtual Ethernet Device	132

1 Introduction

1.1 Overview of eVM for Windows

eVM for Windows provides a virtual machine environment hosting an embedded or real-time operating system alongside Microsoft Windows. eVM for Windows runs in parallel to Windows on any Intel multi-core or multi-threaded PC platform that features Intel Virtualization Technology (VT). Both Windows and the embedded OS run natively, as if they were running on their own dedicated platforms. The result is that, moving an embedded OS plus the application running on it to an eVM platform is a simple process that provides instant access to modern multi-core computing platforms, as well as, access to all of the applications, the development environment, and the rich graphic user interface capabilities that are available under Windows. The key features of eVM for Windows are:

- Memory partitioning such that the guest embedded OS and Windows maintain their integrity – there is no chance for the one operating system to corrupt the other's memory space.
- Memory re-mapping allows both Windows and the guest embedded OS to run native code without any modification to drivers and other memory-mapped utilities.
- Hardware access and interrupt mapping of specific I/O's to the embedded OS, so that they can continue to be served with legacy drivers under the embedded OS.
- Means of emulating legacy I/O's to minimize the rewriting of legacy code. Thus a legacy ISA-based system can easily be migrated to a state of the art and less expensive single-board computer with the simple re-direction of the ISA peripheral to an equivalent on-board PCI device.
- Access to Windows I/O resources by the embedded OS via virtual I/O device drivers.



- ❶ Multi-core Intel Processor platform supporting Virtual technology
- ❷ Un-modified Windows 7, Vista, or XP – running on a dedicated processor Core.
- ❸ Un-modified guest OS – Real-time or other PC compatible embedded OS – running on a dedicated processor Core.
- ❹ eVM for Windows runs on its own, independent of Windows and the guest OS.
- ❺ Virtual I/O device interface allows guest OS to use Windows supported I/O devices.
- ❻ I/O devices that are under the control of Windows **and** Memory that is partitioned for the exclusive access by Windows.
- ❼ I/O devices that are under the control of Windows **and** Memory that is partitioned for the exclusive access by the Guest OS

Figure 1.1 - eVM for Windows

2 Quick Start Guide

It is important to understand the theory behind eVM virtualization before attempting to install and use this product. The next section provides the technical details and background of virtualization and eVM. If you already understand the technical specifics of eVM or just can't wait to get your hands on it, sections 4.1 through 4.4 are quick start guide. Section 4.1 will walk you through the eVM installation process. Section 4.2 will describe installing virtual serial ports. Section 4.2.2 will describe configuring the virtual Ethernet device. Finally, Section 4.3 will guide you through executing the sample guest OS, iRMX, that is provided with the installation.

Important: *If you haven't verified that your target platform has the appropriate VT-x and VT-d features to run eVM for Windows, be aware that the installation runs a verification tool called VtProbe; and the installation will be aborted, if the target platform does meet the virtualization criteria. The stand-alone VtProbe tool (VtProbe.exe) can be downloaded from www.tenasys.com. VtProbe can be used to verify your platform's compatibility with eVM before you begin the product installation.*

3 Product Technical Description

3.1 What is Real-Time embedded virtualization?

Embedded virtualization is a methodology used to partition the hardware resources of a platform between a host operating system and a guest operating system so critical hardware resources can be dedicated to the host operating system. Embedded virtualization is optimized to minimize the virtualization overhead for the guest OSes. Real-time embedded virtualization ensures that critical hardware resources are expressly controlled by the guest OS with minimum, deterministic device latency and minimal CPU overhead.

3.2 Theory of Operation of eVM

The eVM virtualization platform provides a virtual machine that hosts real-time and embedded operating systems running alongside Microsoft Windows. The virtual machine manages the hardware resources to provide a guest OS environment whereby a real-time guest OS can still run as a bounded, real time operating system without modification.

3.2.1 What is eVM

eVM is an embedded virtualization environment that is hosted on the Windows platform to allow consolidation of the real-time control capability of a guest OS with the Windows capability on a common hardware platform.

eVM uses the Windows platform to create a virtual machine capable of supporting a real-time guest OS that can run as a guest OS without degradation in real time performance. eVM is also designed to have minimal impact on the Windows platform, as well. Some embedded products split processing between two computers: one for real-time and the other for Windows interface. Using the eVM virtualization environment allows you to take advantage of both the Windows services and the real-time guest OS services on the same hardware platform. This provides a reduction in hardware cost by not having separate hardware systems, one to support Windows and the other to support the real-time OS. Windows and the guest OS are also tightly connected in the eVM environment, which provides some advantages for communications and interaction between Windows and the real-time guest OS.

3.2.2 Introduction to virtualization

Virtualization is the abstraction of computing resources. In this specific case this abstraction separates the target operating system from its physical resources. When supporting multiple operating systems on the same platform, some interfaces for a given operating system are emulated, and some are mapped directly to a given hardware interface. The emulation of the virtual interfaces is the key to successful virtualization.

Many of Intel's new processor and chipset families have been specifically designed with hardware capabilities that provide virtualization capability and improve the efficiency of the virtual environment through hardware support. These hardware features are known as Intel

Virtualization Technology. Two of the pieces of this Virtualization Technology that are used by eVM are VT-x, Intel Virtualization Technology for processor support, and VT-d, Intel Virtualization Technology for Directed I/O, which is chipset technology.

3.2.3 eVM and Virtualization technology

eVM is designed to take advantage of the Intel Virtualization Technology to run a real-time guest OS without having to modify the RTOS, drivers and interfaces. All that is required is to configure the guest OS for the virtual machine. All important interfaces are handled directly by the guest OS and the applications running on the guest OS, and not through a virtualized layer. This allows the guest OS to run in real-time without modification or customization to the virtual environment.

eVM partitions the platform. Some resources are driven completely by Windows, and some resources are driven by the guest OS. To maximize performance, eVM dedicates a whole CPU hardware thread to the guest OS environment. The hardware thread is not virtualized, in that the dedicated hardware thread is available to run the guest software whenever the guest is ready to run. Devices that are critical to the applications running on the guest OS are dedicated to the guest OS and are directly accessible to the applications running on the guest OS without going through a virtualization layer. The VT-x technology assists the CPU to translate memory addresses in accessing the memory dedicated to the guest OS.

There are also two virtual interfaces for communication between Windows and the guest OS. There is a virtual serial interface and a virtual Ethernet interface that emulate wired links between the operating systems. A single virtual Ethernet link and up to four virtual serial links are supported.

Finally, there are devices on the virtual machine that are completely emulated, because Windows has control of the real PC hardware. Devices such as CMOS, access to the PCI bus, timers, and basic core PC functionality fall into this category.

3.2.4 eVM system requirements (brief overview)

The following lists the hardware requirements for a host platform that can run eVM:

- A multicore Intel processor or a single core processor with hyperthreading.
- Intel Virtualization Technology for x86 CPUs, (Intel VT-x), and a BIOS that supports enabling VT-x.
- The default amount of RAM that will be taken away from Windows and dedicated to the guest OS and eVM is 320 MB. The minimum memory requirement would be 64 MB (for eVM) plus the minimum memory requirement for the guest OS.
- In the case where the guest OS will access a device which uses DMA, it is desirable to have Intel Virtualization Technology for Directed I-O, (Intel VT-d), which eVM will use for DMA remapping. If VT-d is supported by the chipset, then the platform must also have a BIOS that enables VT-d.

It can be difficult to determine which PC platforms and motherboards have VT-x and/or VT-d support. The specific details about the Intel Virtualization Technology support and the BIOS support is not well documented on much of the hardware available, today. It may take some discussions with technical support from the hardware manufacturer, when trying to choose appropriate hardware. It may even require obtaining an evaluation sample for actual testing. TenAsys provides a tool, VtProbe, which checks a given platform for VT-x and VT-d support, see Appendix D.

3.2.5 Performance: How eVM addresses RTOS / Windows Application Needs

As we said earlier, the goal of eVM is to provide a virtual environment, hosted on a Windows platform, that looks and performs like a dedicated PC for the guest OS. The following discusses the performance issues to be aware of.

To provide bounded, real-time performance for the guest OS, the design of eVM is such that interrupts are delivered to the guest RTOS in order to be handled in a deterministic way. This is accomplished, in part, by making sure that real-time interrupts are not handled by Windows, and Windows is prevented from ever masking a real-time interrupt. The other part of deterministic interrupt handling is the design of the interrupt forwarding mechanism which is optimized to reduce interrupt latency and jitter and to provide a direct handoff to the real-time guest OS.

Hardware performance of the guest OS is maximized by providing a dedicated hardware thread for the guest OS. The guest does not have to share that core or hardware thread with any other operating system.

Since the guest OS has a dedicated hardware thread, Windows has its own dedicated hardware threads. Thus, Windows performance is not reduced by having to share CPU resources. Windows interrupts are handled directly by Windows. Aside from removing one core and some memory from the Windows system, there is minimal impact to Windows normal operation.

3.2.6 Communications between Windows and the Guest: Virtual Device Interfaces

When discussing the performance aspects of eVM, the isolation between Windows and the guest OS has been emphasized, but there will be times when both OSes will want to communicate with one another and share data. With both OSes on the same hardware, there are some unique opportunities to provide efficient communication mechanisms between the OSes, for example, there is a Virtual Ethernet connection that provides communication between the guest OS and Windows. This has an NDIS driver on the Windows side and an NE2000 interface on the guest OS side. This virtual interface uses shared memory to move packets between Windows and the guest OS as if there were a wired Ethernet link between the two OSes. There are also virtual serial ports that provide data interchange between Windows and the guest OS via a standard serial port interface.

3.2.7 eVM Architecture: Partitioning the PC between Windows and Guest OS

Let's take a look at how the PC platform is partitioned using eVM to separate Windows and a guest OS.

3.2.7.1 Hardware Threads

On installation, one hardware thread is removed from Windows control and dedicated to the Guest OS. This uses standard Windows configuration mechanisms to manage this. In a multicore platform, this will dedicate a core to eVM and the guest OS.

3.2.7.2 Memory

On installation, the memory required for eVM and the guest OS is removed from the Windows memory pool and dedicated to the guest OS using similar standard Windows configuration mechanisms as for the thread control. This memory will be non-paged, because it's not under Windows control anymore. The memory that is dedicated to eVM and the guest OS is allocated before Windows boots.

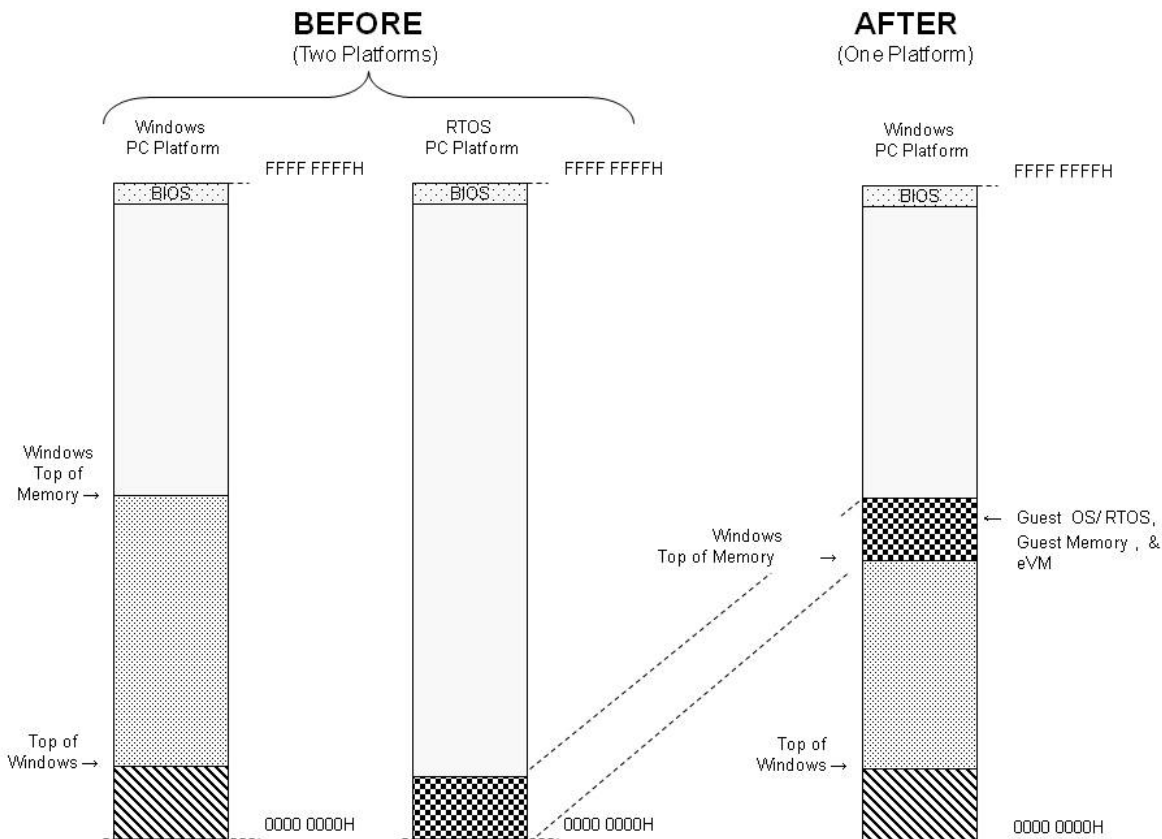


Figure 3.1 - Memory Map

All memory addresses have to be translated to appear to be addressed from address 0 of the virtual PC, so it looks like a real dedicated PC. VT-x assists this memory remapping in hardware, thus there is no modification of the guest OS required.

3.2.7.3 Interrupts

In order to access a device directly from the guest OS, it is necessary to dedicate any interrupt resource associated with that device to eVM, so eVM can manage it and not Windows. eVM maintains the control of these resources and then passes control to the Guest OS. With eVM, not Windows managing these interrupt resources, the guest OS maintains a deterministic system environment.

3.2.7.4 Devices

Virtual device is one class of device: disc controllers, CMOS, PCI bus, host bridge on the PCI bus, timers, and interrupt controller. These devices are created in software and presented to the guest OS as if they were real hardware devices. These virtual devices are all part of the virtual machine.

A second class of devices is of real devices that are assigned to the guest OS and taken away from Windows. The guest OS has exclusive access to these devices and does not share them with Windows. A dedicated PCIe Ethernet card is an example of this. PCI or PCIe devices can be dedicated to the guest OS such that an application specific device is managed by the RTOS and Windows is not aware of the device.

3.2.8 Resources that are not Isolated between Windows and the Guest OS

Per the virtualization architecture, eVM provides isolation between Windows and the Guest OS, but there are shared resources that cannot be managed in software and the usage of these resources by one of the OSes can impact the other OS. The memory controller, internal graphics controller (also uses the memory controller), and the front side bus are examples of these kinds of resources. For example, heavy usage of the memory controller by one of the OSes is going to impact performance on the whole platform. The selection of processor, chipset, and clocking will have an influence on the overall system performance. For the very highest performance applications, the choice of hardware is going to have to be done carefully and evaluated with the appropriate measurements to determine the optimum hardware platform.

3.2.9 Device Management

The eVM Device Manager utility is used for the management of devices that will be dedicated to that guest OS. Before using the eVM Device Manager to manage devices, it is important to know what devices can be dedicated to the guest OS, what devices cannot, and why.

3.2.9.1 What is a device

The devices that eVM can dedicate to a guest OS are PCI devices, PCI Express (PCIe) devices, and legacy devices (COM, LPT, etc). These devices come in a variety of flavors; so it is important to determine if the device can, in fact, be dedicated to a guest OS. The hardware features that will be used to determine the device's suitability for being a dedicated, guest OS, device are:

1. Device Access: Does the device use memory mapped registers or I/O mapped registers?
2. Interrupt capability: Does the device generate an interrupt; and if so, what type of interrupt does it generate?
3. Bus Master capability: Is the device a bus master device, i.e. does the device generate its own memory accesses?

An example of a device that cannot be dedicated to the guest OS would be a USB memory disk. That is because the USB memory disk interfaces through the USB controller. It is not interfaced directly to the bus nor is it accessed directly by the device driver.

3.2.9.2 Identifying your device's capabilities

The following describes the 3 classes of device capability that are of interest when configuring devices in eVM for dedicated access by the guest OS.

1. Device Access:

The method of device access to the device registers must be determined. The device can either use the I/O space or be memory mapped. The CPU Virtualization Technology, VT-x, can manage the mapping of direct memory or I/O space for the virtual machine. The VT-x mapping also provides memory protection. Any access to a memory or I/O space that is not mapped would be trapped by eVM and blocked. Thus errant driver software in the guest OS could not corrupt the Windows platform.

Legacy devices must be explicitly configured. Their I/O ranges will have to be determined and configured in eVM, so that eVM can allow access to these I/O ports by the guest OS.

PCI and PCIe devices can be discovered. All the PCI/PCIe register and memory access information is contained in the PCI header. eVM reads the Windows configuration for the device and sets up the device's registers and memory to be transparently accessible from the guest OS. No interaction is necessary by the user to configure this device after it has been assigned to eVM.

2. Interrupt Capability:

The interrupt capability of the device needs to be determined. If the device has interrupt capability, then it needs to be determined if the interrupt will be used by the guest OS. eVM then needs to be appropriately configured.

3. Bus Master Capability

Finally, it needs to be determined if the device is a bus master device, i.e. does the device contain a DMA engine and can it generate its own memory accesses.

3.2.9.3 *Interrupts*

There are two classes of interrupts. The first class is the legacy interrupts, or IRQs that have the device interrupt lines connected to specific inputs of an interrupt controller, and the interrupt controller manages how each interrupt, IRQ, is delivered to the CPU. The second class is the more modern Message Signaled Interrupt (MSI), where the device performs a special bus write to signal the chipset. The device has no interrupt lines. Instead it has special parameters that the device writes to a memory space reserved for MSI interrupt activity. When the device writes to the MSI memory space, the interrupt is delivered directly to the CPU.

eVM will handle both classes of interrupts for devices that are dedicated to the guest OS. All dedicated device interrupts are fielded by eVM. Then eVM, in turn, forwards those interrupts to the guest OS when it is ready to receive interrupts. The virtual machine provides virtual interrupt services for the guest OS. The guest OS may mask and unmask interrupts in the virtual machine. eVM monitors this, and holds off interrupt forwarding for interrupts that are masked in the virtual machine, and then forwards those interrupts to the virtual machine when they are unmasked.

3.2.9.4 *Software Considerations*

To provide a PC-like virtual machine, eVM must manage both classes of interrupt and act as an interrupt controller for the virtual machine. The guest OS interacts with the virtual interrupt controller of eVM as if it were a hardware interrupt controller. The eVM virtual interrupt controller provides interrupt signaling to the guest OS, interrupt masking capability, and interrupt enabling and disabling. Therefore, if the guest has interrupts disabled, then eVM must wait until the interrupts are enabled before passing the interrupt to the guest OS.

The virtual machine that supports the guest OS has a different interrupt mapping for the virtual machine than the actual hardware platform. The virtual machine presents an 8259 legacy interrupt controller as the means for interfacing the dedicated device interrupts to the guest OS. eVM maps both classes of interrupts, legacy and MSI, as IRQ's in the virtual machine. The guest OS sees all interrupts as legacy IRQ interrupts in the virtual machine irrespective of whether the real hardware interrupt was a legacy interrupt or an MSI interrupt. eVM is acting like a software interrupt controller. Specific mapping of devices to IRQs in the virtual machine is described by the Virtual Machine specification, see Appendix B.

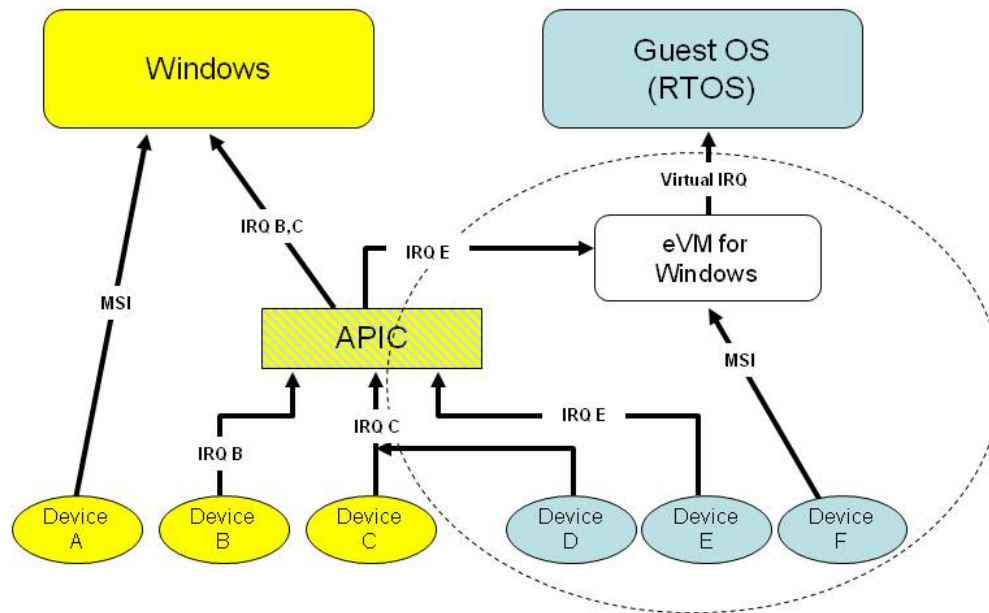


Figure 3.2 - eVM Interrupt Mapping

Device A: Is a PCI device and the driver supports MSI – *Note: Windows XP and earlier versions of Windows do not support MSI.*

Device B: Is handled normally by Windows.

Device C and D: Devices share the same IRQ line and the RTOS needs to access the Device D, so the device is passed to the RTOS but not the IRQ. Device C remains a fully functional Windows device. Device D may be polled by the Guest OS, but it cannot deliver an interrupt to the guest OS.

Device E: Is passed on to the guest OS. IRQ E is serviced by eVM then delivered to the RTOS as a virtual IRQ.

Device F: Is an MSI device. The MSI is handled by eVM and delivered to the RTOS as a virtual IRQ.

Note: eVM always handles the IRQ, meaning that eVM's interrupt handler runs when the interrupt fires. The eVM interrupt handler converts the interrupt to an input for the virtual Peripheral Interrupt Controller (PIC) and forwards the translated vector to the guest OS.

Example: Ethernet controller capture

If an Ethernet controller that uses IRQ18 is to be captured and dedicated to the virtual machine, and hence, the guest OS, the first thing that eVM will do is install an interrupt handler for IRQ18 to manage the real interrupt on the hardware. Next, eVM will map that interrupt into one of the virtual machine's IRQs, a different IRQ from the actual hardware interrupt. Then in operation, the eVM interrupt handler will handle IRQ18. It will map IRQ18 to an available IRQ on the virtual machine, and it will forward the interrupt to the guest OS when the guest OS is ready to handle it.

There is some additional latency introduced by the forwarding and mapping of the real hardware interrupt to the interrupt on the virtual machine, but this latency is fixed and bounded. Because this latency is fixed and bounded, the guest OS can still run in a deterministic fashion. eVM has been optimized to minimize this latency, but this latency is also determined by the hardware. Some CPU, chipset, and board designs are faster than others, but the hardware contribution to this latency will be fixed. The virtual machine will still be deterministic.

3.2.9.5 System hardware considerations – what eVM can or can't do!

There are some restrictions to what can and cannot be done when mapping interrupts to the virtual machine and still maintain a deterministic system. Legacy interrupts from PCI devices present a restriction, because multiple PCI devices can share a single interrupt. In fact, PCI is designed to share an interrupt. This interrupt sharing presents a problem, if the devices that are sharing an interrupt are not configured to be used by the same OS. That is to say, if one device is to be used as a Windows device and another device is to be used as a guest OS device and they both share the same interrupt, then there is no way to provide deterministic response for the guest device, since it is dependent on a Windows interrupt handler running.

When an interrupt occurs on a shared interrupt, the interrupt handlers for all devices sharing that interrupt must be run to determine which device caused the interrupt and to clear the interrupt so it will be available to handle a subsequent event. This would mean that the guest OS would be dependent on a Windows interrupt handler running to maintain a deterministic system. Therefore, it is not allowed to assign a device to the guest and a device to Windows that share a common interrupt. The eVM Device Manager will flag an attempt to do this as a conflict. A conflict of this type is a property of the motherboard and/or chipset design. There is no way to configure around that, so it is important to choose a hardware design that will allow interrupt isolation of devices that are to be dedicated to the guest OS. One alternative for a device that would have an interrupt conflict with another device used by Windows would be to run the device in a polled mode. The device would be configured with interrupts turned off and register access can still be dedicated to the guest OS. It should be noted that MSI interrupts do not have this problem.

3.2.9.6 DMA

A DMA device can generate its own memory accesses without CPU intervention. Such a device is a bus master device. For bus master devices, the Intel Virtualization Technology, VT-x, plays no part. VT-x manages address remapping for the CPU. Addresses that the DMA engine of a bus master device would use must be remapped for the virtual machine. The view of the memory that is assigned to the virtual machine starts at address zero, but the actual physical address of that memory is offset from memory address zero an amount dependent on the system configuration. Any addresses that the DMA engine would use must be mapped to the actual physical address of the memory, so the physical memory offset must be added to the virtual memory addresses of the virtual machine.

This DMA address mapping can be handled in two ways. One way is to modify the device driver so that it will do the DMA address mapping. This requires that a modified device driver must be prepared and installed on the guest OS. The modified device driver must be able to determine the memory offset of the virtual machine's physical memory and apply that offset to any DMA operation, see Appendix E. The other method for DMA address mapping is to use hardware that is specifically designed to provide address mapping of bus master devices. Intel Virtualization Technology, VT-d or Virtualization for Directed I/O, is the hardware mechanism that is available for DMA address mapping. The VT-d technology resides in specific chipsets that contain a DMA redirection unit that translates bus master address accesses on the fly.

3.2.9.7 Hardware consideration – VT-d

eVM defines an address domain that is assigned to the virtual machine for the guest OS. Any device assigned to eVM is also assigned to that memory domain in hardware. eVM will then translate any address that is generated in that domain and map it to the physical address of the virtual machine domain. If a device tries to access memory outside its domain, it is blocked by eVM, providing memory security and hardware protection, as well as, memory mapping.

If VT-d is not present, a standard bus master driver will not work and will compromise Windows, because a bus master device that is operating with a device driver that has not been modified for the virtual machine, will address the wrong area of memory, and almost certainly corrupt Windows memory. The bus master driver must be modified to work with the virtual machine and do the appropriate address translations to be able to function in the virtual machine if VT-d is not present. Therefore, it is highly desirable to have VT-d supported by the hardware and enabled through the BIOS, if bus master devices are to be dedicated to the virtual machine, and hence, the guest OS.

3.2.10 What Happens when a Device is assigned to eVM

When the eVM Device Manager is used to assign a device to eVM from Windows, the following actions are taken:

- Access to the device by Windows is stopped by removing the Windows device driver for that device.

- Device identity information is passed from Windows to eVM, so eVM can recognize it. Thus when eVM next starts up, it gets notified that it now owns this device.
- Device identity can be passed to eVM with its IRQ or without its IRQ. If the device is passed with its IRQ, then a check is made for an interrupt sharing violation. If there is an interrupt sharing violation, the eVM Device Manager will post a warning.
- When eVM next starts and sees that this device now belongs to eVM, it obtains the device identifier and discovers the device. If the device has an IRQ, eVM will assign an interrupt handler for it. If it is PCI device, eVM inserts the device into one of the virtual PCI slots that are provided by the virtual machine. The virtual machine that eVM creates has 5 virtual PCI slots for dedicated PCI devices.
- If the device is a PCI device, eVM Scans the base address registers for that device and configures the virtual machine so the access to those base address registers go directly to that device and are not intercepted by eVM.
- If the device has an interrupt, eVM assigns a virtual IRQ in the virtual machine's interrupt map.
- If the device does not have an IRQ, eVM will disable the interrupt in the virtual PCI device header of the virtual machine.
- If this device is capable of being a bus master and if VT-d is enabled, then eVM assigns the device to the VT-d domain of the virtual machine for hardware address translation in the guest OS.
- If this device is capable of being a bus master but VT-d is not present, a warning will be presented by the eVM Device Manager. It is the responsibility of the developer to either provide the modified driver or disable bus master capability.

4 Installation

4.1 Installation of eVM for Windows

The following provides a step-by-step guide of the installation screens with comments as necessary, including Windows and eVM setup and License activation. The following description is for installation on Windows 7/Vista. ***If you are installing on Windows XP, following the step-by-step guide in Appendix F.***

1. Run **evm100-install.exe** which begins the installation process by installing the Microsoft Visual C++ 2008 Redistributable, see Figure 4.1.

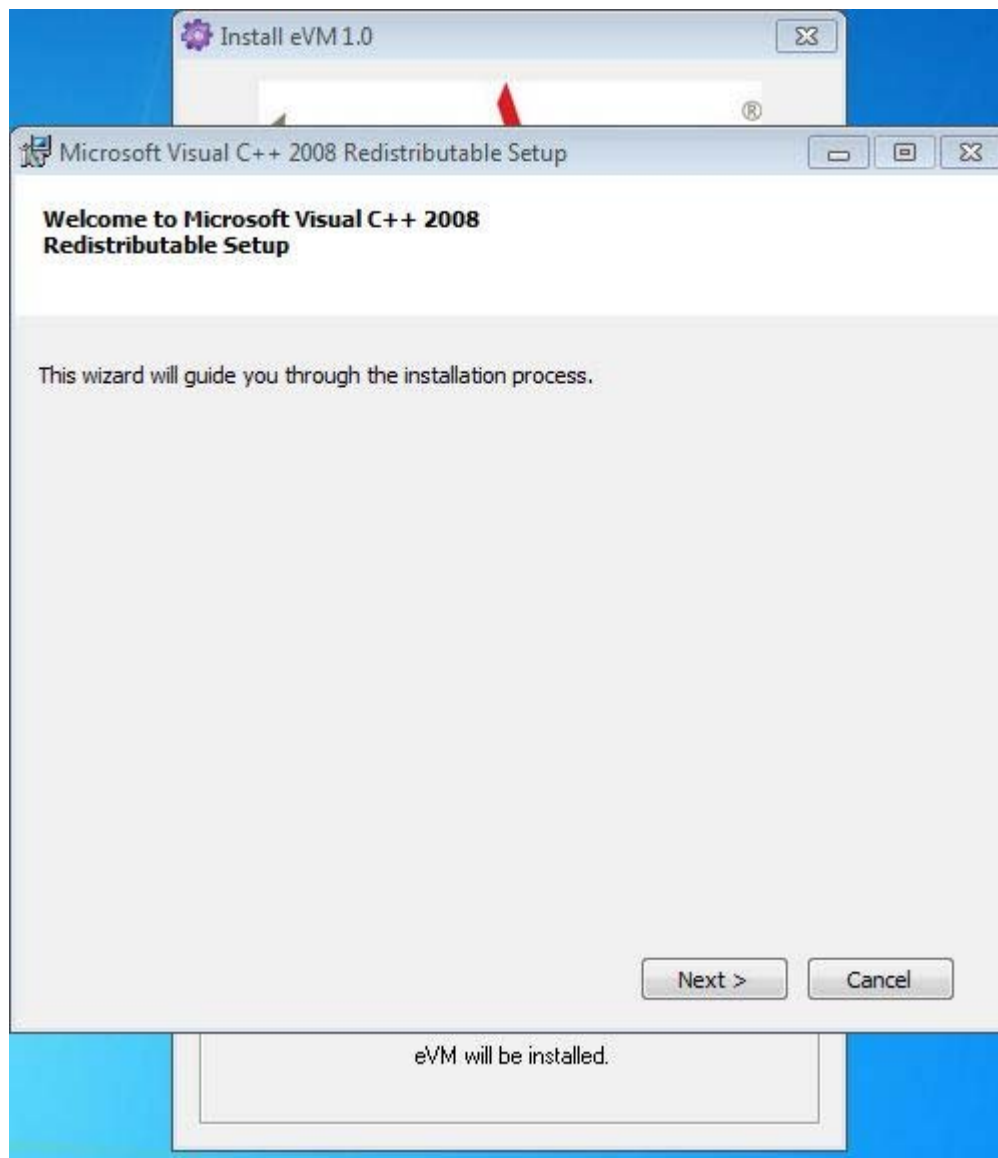


Figure 4.1 - Installing Visual C++ 2008 Redistributable

2. Accept the Microsoft Visual C++ 2008 Redistributable license by checking the checkbox, see Figure 4.2. Then click **Install**.

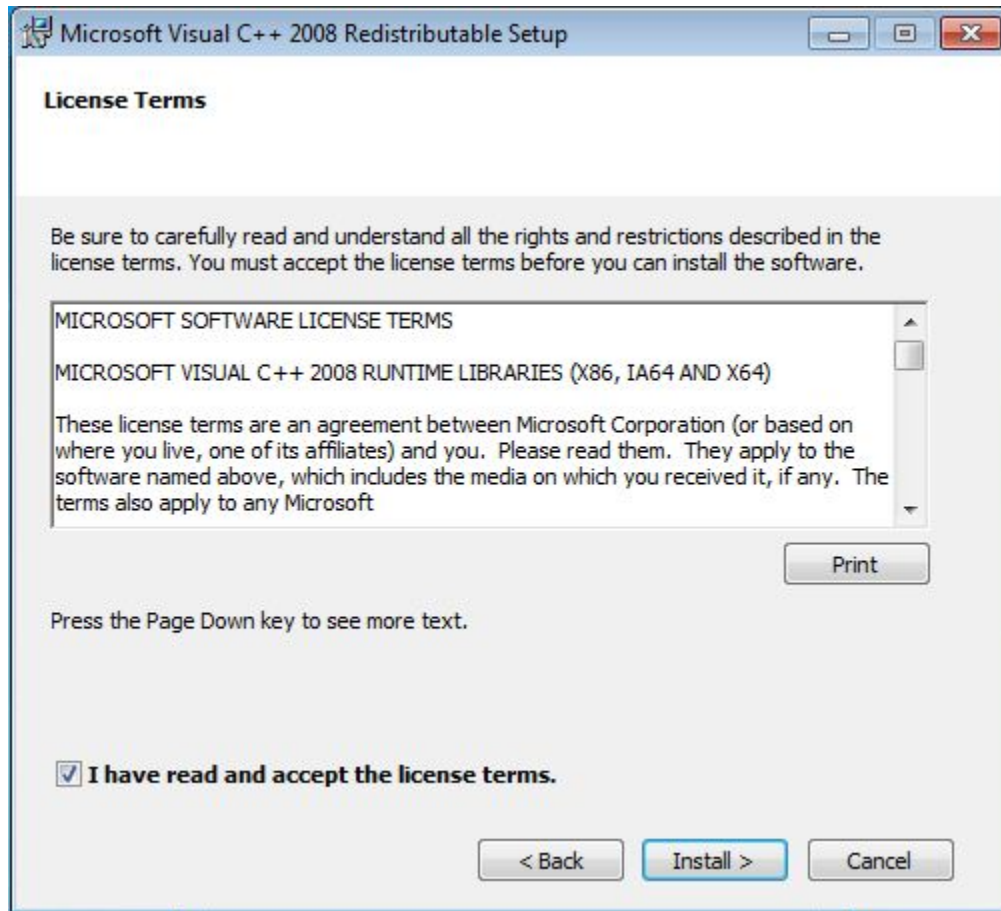


Figure 4.2 - Accept the Microsoft Visual C++ 2008 Redistributable License

3. A progress dialog will show the redistributable installation and setup. When it is complete, the following dialog in Figure 4.3 will be displayed. Click **Finish** to continue with the installation.

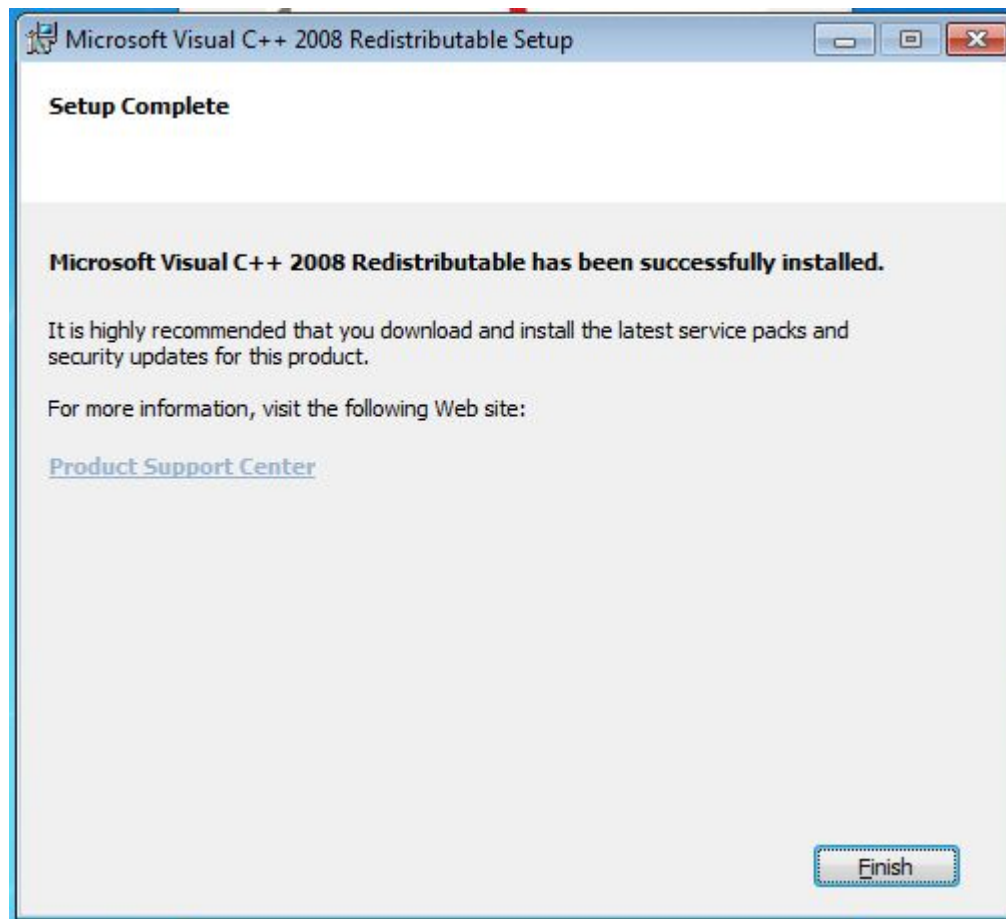


Figure 4.3 - Visual C++ 2008 Redistributable Setup Complete

4. With the Visual C++ 2008 Redistributable installation complete, the eVM Installation Wizard is automatically launched, see Figure 4.4. Click **Next**.



Figure 4.4 - eVM Installation Wizard Starting

5. Read and accept the TenAsys eVM Software License Agreement; and click **Next** to continue, see Figure 4.5

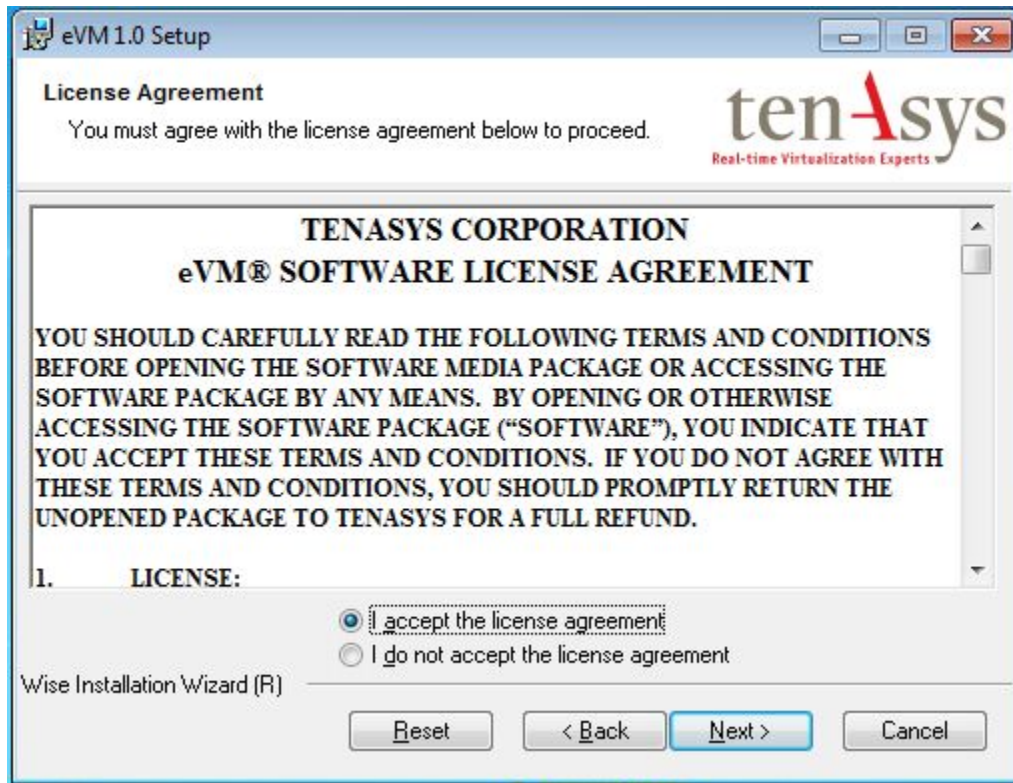


Figure 4.5 - TenAsys eVM Software License Agreement

6. Select the folder where you would like the installer to put the eVM installation files. The default location is "C:\Program Files\eVM\", see Figure 4.6. Then click **Next** to continue.



Figure 4.6 - Select eVM Installation Folder

7. Select the type of installation that you would like. If you choose *Standard Installation*, Figure 4.7, eVM will be installed with virtual port support. If you choose *Complete Installation*, Figure 4.8, eVM with virtual port support, all the utilities, and a sample guest OS, iRMX will be installed. You can also choose a custom installation. After selecting the installation type, click **Next** to continue.

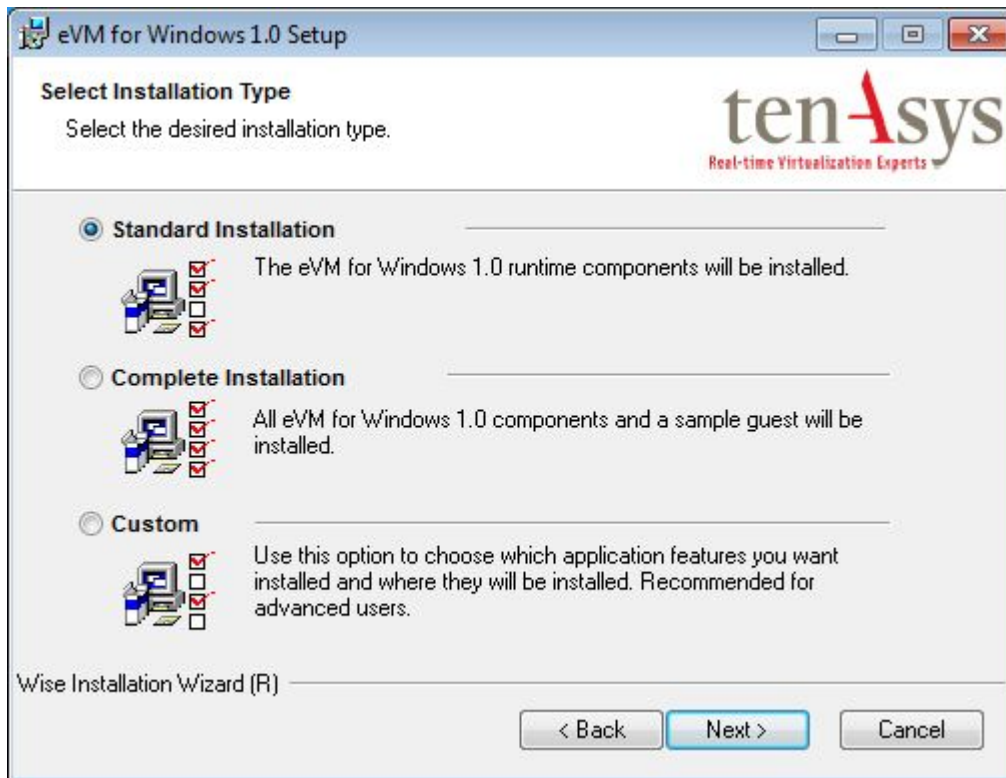


Figure 4.7 - Choosing Standard Installation Type

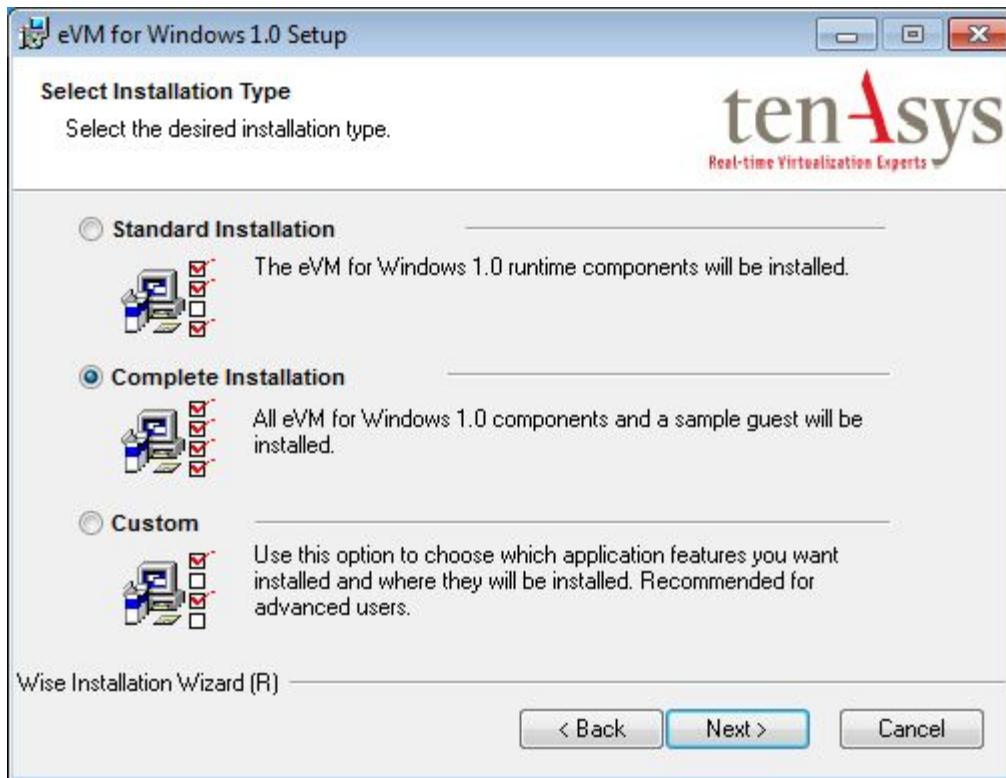
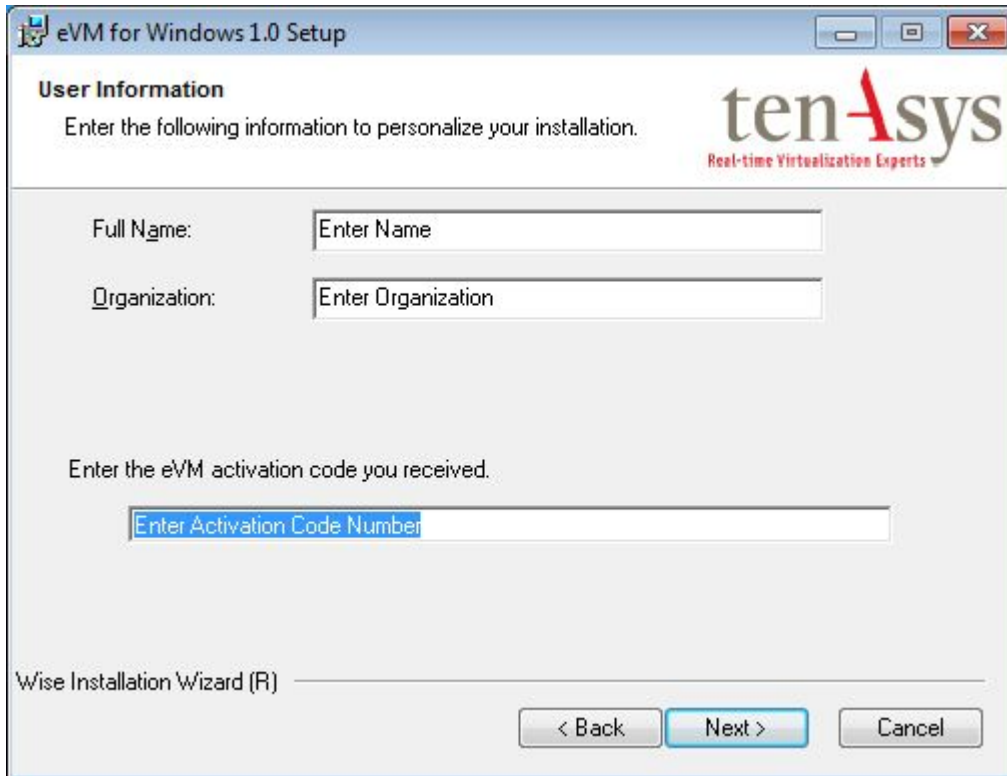


Figure 4.8 – Choosing Complete Installation Type

8. The eVM Installation Wizard will now prompt you to enter your name, organization, and your eVM activation code that you received when you purchased and downloaded eVM, see Figure 4.9. Make sure you fill out all the fields properly and **be sure that you are connected to the Internet, as the activation code will be verified with TenAsys over the Internet.** Then Click **Next** to proceed with the installation.



The screenshot shows a Windows-style window titled "eVM for Windows 1.0 Setup". Inside, the "User Information" section is active, with the instruction "Enter the following information to personalize your installation." and the TenAsys logo. There are three input fields: "Full Name:" with a placeholder "Enter Name", "Organization:" with a placeholder "Enter Organization", and a larger field for the "eVM activation code" with a placeholder "Enter Activation Code Number". At the bottom, there are three buttons: "< Back", "Next >" (highlighted in blue), and "Cancel". The status bar at the bottom left reads "Wise Installation Wizard (R)".

Figure 4.9 - eVM Activation

9. When your activation code has been verified, the eVM Installation Wizard will then prompt you to confirm that you want to proceed with the installation, Figure 4.10. If you want to check or change anything at this point, you can click the **Back** button.

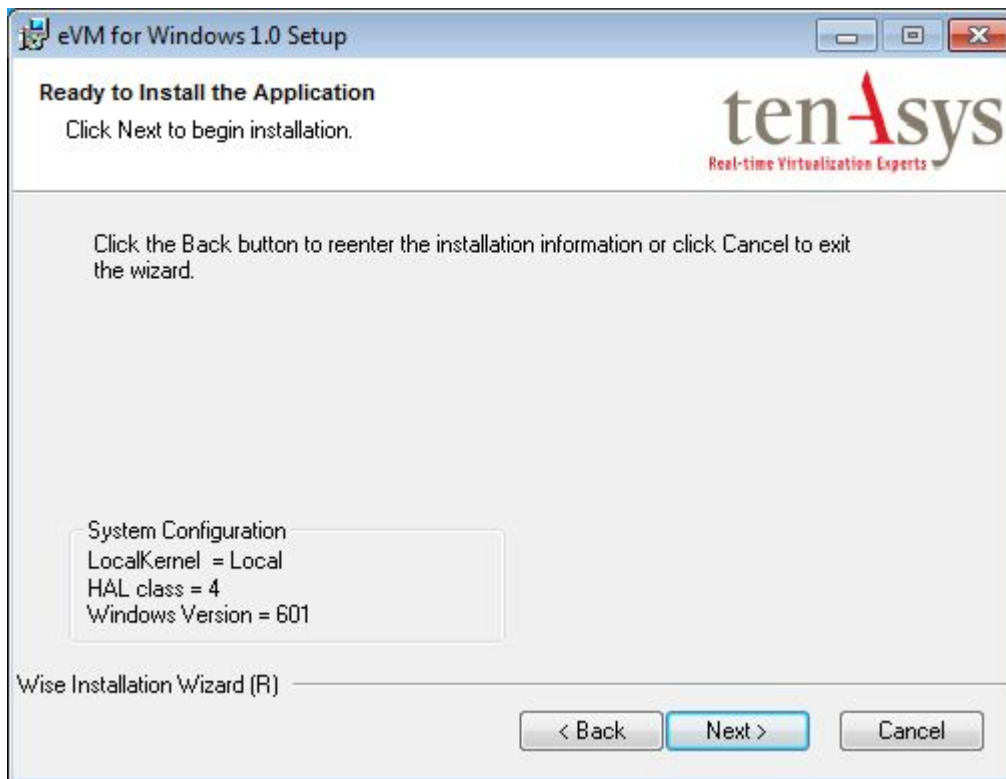


Figure 4.10 – eVM Installation Verification

To proceed with the installation, click **Next**.

10. You will then see a progress dialog, see Figure 4.11, the will show you the status of the components and features that are being installed.

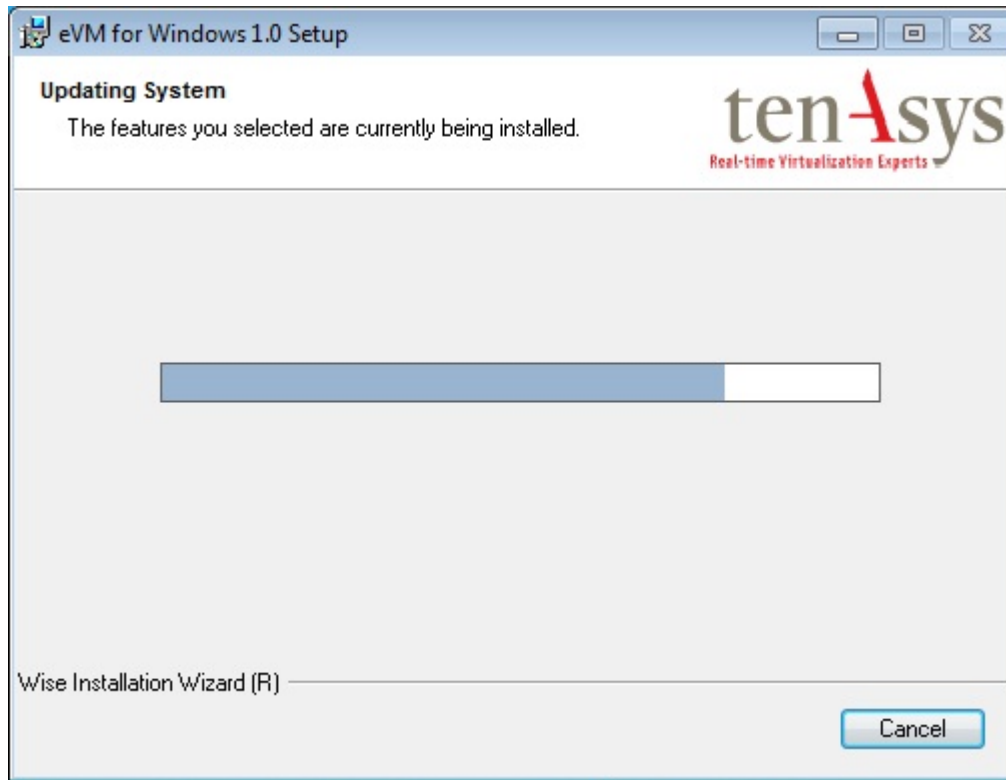


Figure 4.11 – eVM Installation Wizard Progress Dialog

11. When all components and features of eVM that you selected have been installed, the Wizard will indicate that the installation was completed successfully, see Figure 4.12. If you want to view the installation log to see what the Wizard did, check the **checkbox** to view the log. Whether you view the log or not, note the path where the log is saved. If you experience difficulties with the installation that you cannot resolve, TenAsys technical support may ask to see this log to help determine the problem. Now click **Finish**.



Figure 4.12 - eVM Installation Wizard Successful Installation

12. If you chose to view the installation log, you will see something similar to Figure 4.13. After viewing the log, you can close Notepad.

```

File Edit Format View Help
===== verbose logging started: 2/1/2010 22:52:34 Build type: SHIP UNICODE 5.00.7600.00 Calling process: C:\windows
MSI (c) (68:30) [22:52:34:504]: Font created. Charset: Req=0, Ret=0, Font: Req=MS Shell Dlg, Ret=MS Shell Dlg
MSI (c) (68:30) [22:52:34:504]: Font created. Charset: Req=0, Ret=0, Font: Req=MS Shell Dlg, Ret=MS Shell Dlg
MSI (c) (68:E8) [22:52:34:504]: Resetting cached policy values
MSI (c) (68:E8) [22:52:34:504]: Machine policy value 'Debug' is 0
MSI (c) (68:E8) [22:52:34:504]: ***** RunEngine:
*****
***** Product: C:\Users\ \AppData\Local\Temp\evm100.msi
*****
***** Action: *****
***** Command Line: *****
MSI (c) (68:E8) [22:52:34:504]: Machine policy value 'DisableUserInstalls' is 0
MSI (c) (68:E8) [22:52:34:504]: SOFTWARE RESTRICTION POLICY: verifying package --> 'C:\Users\ \AppData\Local\
MSI (c) (68:E8) [22:52:34:504]: Note: 1: 2262 2: DigitalSignature 3: -2147287038
MSI (c) (68:E8) [22:52:34:504]: SOFTWARE RESTRICTION POLICY: C:\Users\ \AppData\Local\Temp\evm100.msi is not
MSI (c) (68:E8) [22:52:34:504]: SOFTWARE RESTRICTION POLICY: C:\Users\ \AppData\Local\Temp\evm100.msi is perm
MSI (c) (68:E8) [22:52:34:504]: Cloaking enabled.
MSI (c) (68:E8) [22:52:34:504]: Attempting to enable all disabled privileges before calling Install on Server
MSI (c) (68:E8) [22:52:34:519]: End dialog not enabled
MSI (c) (68:E8) [22:52:34:519]: Original package ==> C:\Users\ \AppData\Local\Temp\evm100.msi
MSI (c) (68:E8) [22:52:34:519]: Package we're running from ==> C:\Users\ \AppData\Local\Temp\evm100.msi
MSI (c) (68:E8) [22:52:34:519]: APPCOMPAT: Compatibility mode property overrides found.
MSI (c) (68:E8) [22:52:34:519]: APPCOMPAT: looking for appcompat database entry with ProductCode '{E713FEDC-4C78-4B
MSI (c) (68:E8) [22:52:34:519]: APPCOMPAT: no matching ProductCode found in database.
MSI (c) (68:E8) [22:52:34:519]: MSCOREEE not loaded loading copy from system32
MSI (c) (68:E8) [22:52:34:519]: Machine policy value 'TransformsSecure' is 0
MSI (c) (68:E8) [22:52:34:519]: User policy value 'TransformsAtSource' is 0
MSI (c) (68:E8) [22:52:34:519]: Machine policy value 'DisablePatch' is 0
MSI (c) (68:E8) [22:52:34:519]: Machine policy value 'AllowLockdownPatch' is 0
MSI (c) (68:E8) [22:52:34:519]: Machine policy value 'DisableLUAPatching' is 0
MSI (c) (68:E8) [22:52:34:519]: Machine policy value 'DisableFlyweightPatching' is 0
MSI (c) (68:E8) [22:52:34:519]: Enabling baseline caching for this transaction since all active patches are MSI 3.0
MSI (c) (68:E8) [22:52:34:519]: APPCOMPAT: looking for appcompat database entry with ProductCode '{E713FEDC-4C78-4B
MSI (c) (68:E8) [22:52:34:519]: APPCOMPAT: no matching ProductCode found in database.
MSI (c) (68:E8) [22:52:34:519]: Transforms are not secure.
MSI (c) (68:E8) [22:52:34:519]: PROPERTY CHANGE: Adding MsiLogFileLocation property. Its value is 'C:\Users\
MSI (c) (68:E8) [22:52:34:519]: Command Line: WISE_SETUP_EXE_PATH=C:\Users\ \AppData\Local\Temp\evm100.exe CU
MSI (c) (68:E8) [22:52:34:519]: PROPERTY CHANGE: Adding PackageCode property. Its value is '{620342AB-AE80-49A7-BC3
MSI (c) (68:E8) [22:52:34:519]: Product Code passed to Engine.Initialize:
MSI (c) (68:E8) [22:52:34:519]: Product Code from property table before transforms: '{E713FEDC-4C78-4BDE-8E91-77D9B
MSI (c) (68:E8) [22:52:34:519]: Product Code from property table after transforms: '{E713FEDC-4C78-4BDE-8E91-77D9B
MSI (c) (68:E8) [22:52:34:519]: Product not registered: beginning first-time install
MSI (c) (68:E8) [22:52:34:519]: PROPERTY CHANGE: Adding ProductState property. Its value is '-1'.
MSI (c) (68:E8) [22:52:34:519]: Entering CmsiConfigurationManager::SetLastUsedSource.
MSI (c) (68:E8) [22:52:34:519]: User policy value 'SearchOrder' is 'nmu'
MSI (c) (68:E8) [22:52:34:519]: Adding new sources is allowed.
MSI (c) (68:E8) [22:52:34:519]: PROPERTY CHANGE: Adding PackageCodeChanging property. Its value is '1'.
MSI (c) (68:E8) [22:52:34:519]: Package name extracted from package path: 'evm100.msi'
MSI (c) (68:E8) [22:52:34:519]: Package to be registered: 'evm100.msi'
MSI (c) (68:E8) [22:52:34:535]: Note: 1: 2262 2: AdminProperties 3: -2147287038
MSI (c) (68:E8) [22:52:34:535]: Machine policy value 'DisableMsi' is 0
  
```

Figure 4.13 - eVM Installation Wizard Installation Log

13. After you click **Finish** in the dialog shown in Figure 4.12, the Wizard will prompt you that you now need to restart your system, see Figure 4.14. A restart is required, so eVM can take the memory that it needs for eVM and the guest OS from Windows. In general, resizing Windows available memory requires a restart of the Windows OS. Click **Yes**, and wait for the system to reboot.

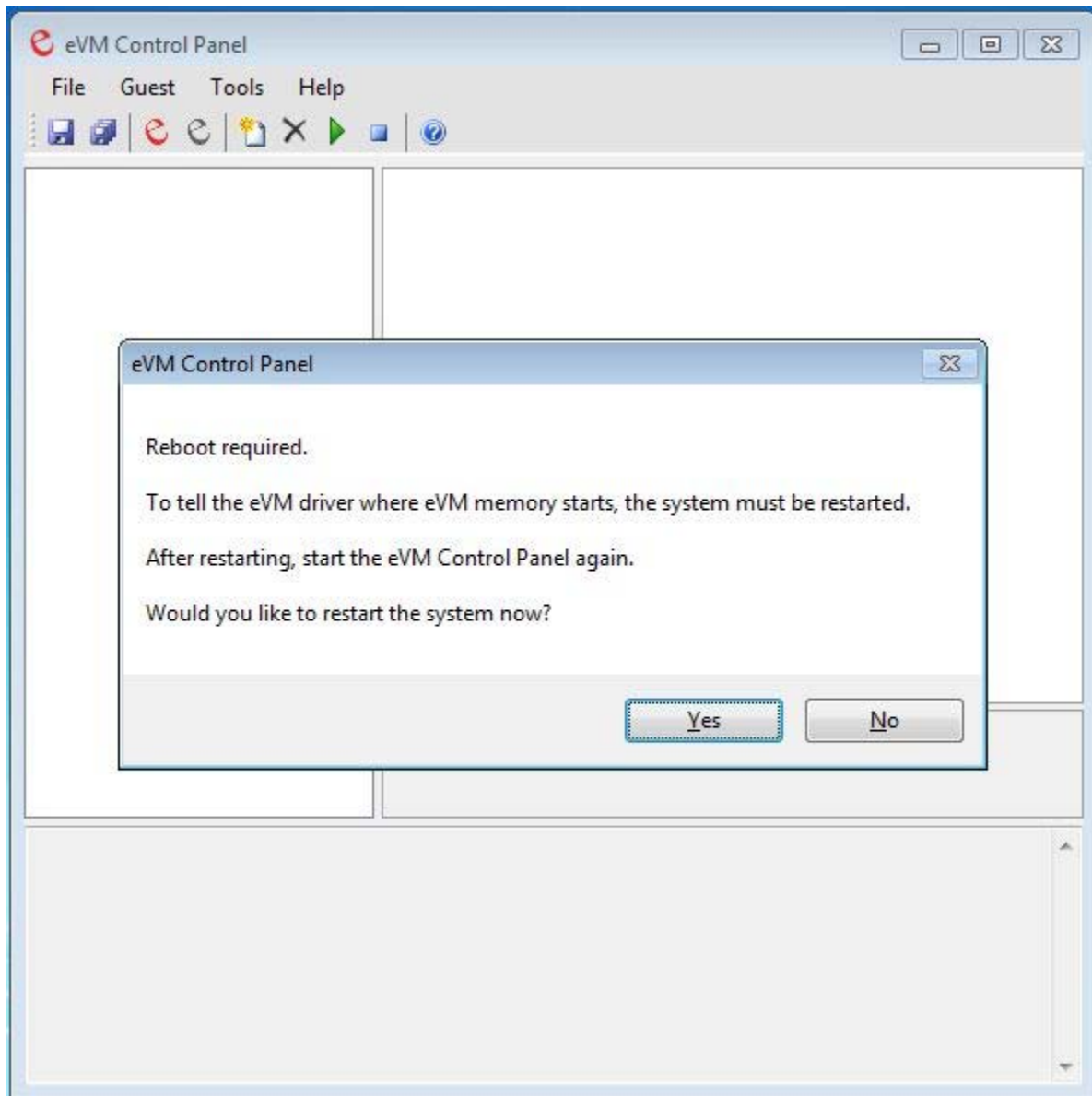


Figure 4.14 - eVM Control Panel Restart Prompt

14. After the system has rebooted, wait for the eVM services to start, which will be indicated by the 'e' icon showing up in the system tray, see Figure 4.15.



Figure 4.15 - eVM System Tray Icon

15. Click on the **eVM tray icon** and select **eVM Control Panel** to launch the eVM Control Panel, see Figure 4.16.

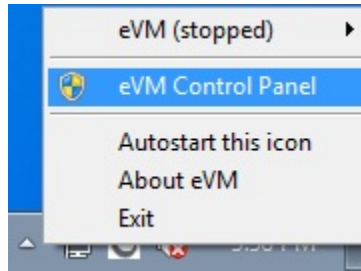


Figure 4.16 - Launching the eVM Control Panel from the system tray

16. When the eVM Control Panel launches, if you chose the *Standard Installation*, you will see the system configuration and no eVM guests installed, Figure 4.17. If you look at the configuration in the right pane, 2 virtual serial ports have been installed with the option of *next_free* selected.

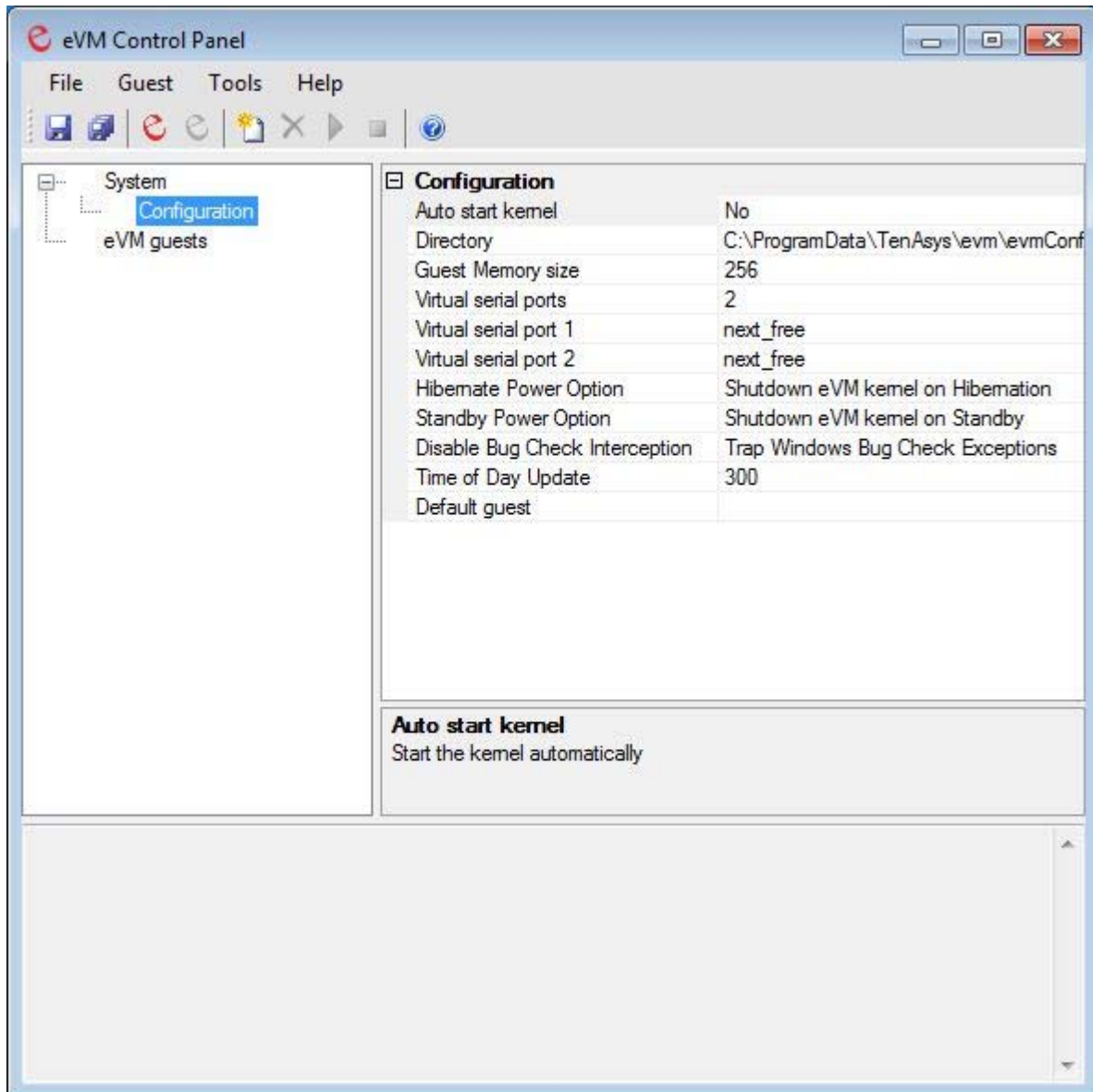


Figure 4.17 - eVM Control Panel with iRMX Guest RTOS Installed

If you chose the *Complete Installation*, not only has eVM been installed with the 2 virtual serial ports, Figure 4.18, but also has the sample guest RTOS, iRMX, complete with a sample configuration been installed, Figure 4.19.

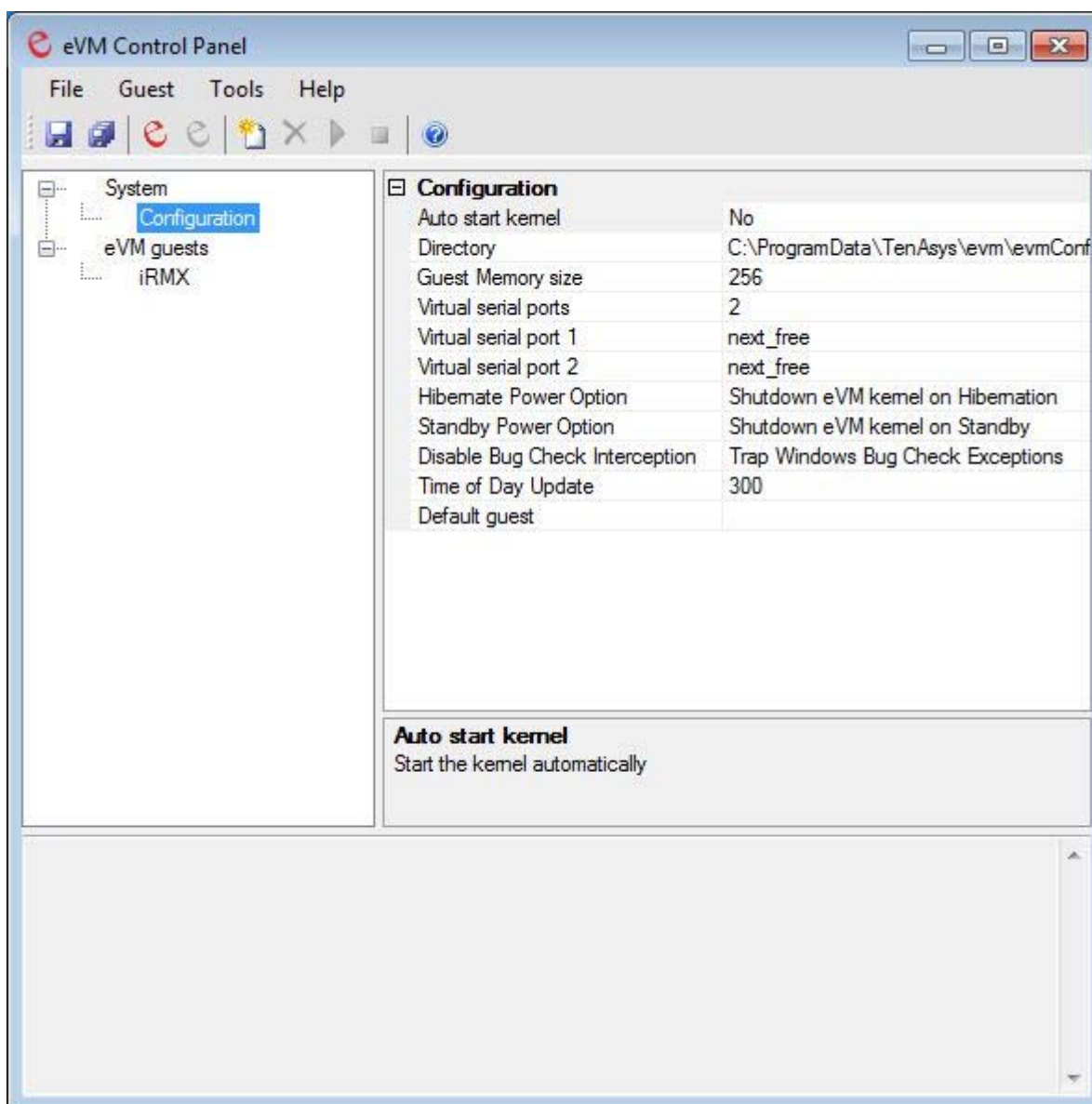


Figure 4.18 - Complete eVM Installation Showing System Configuration

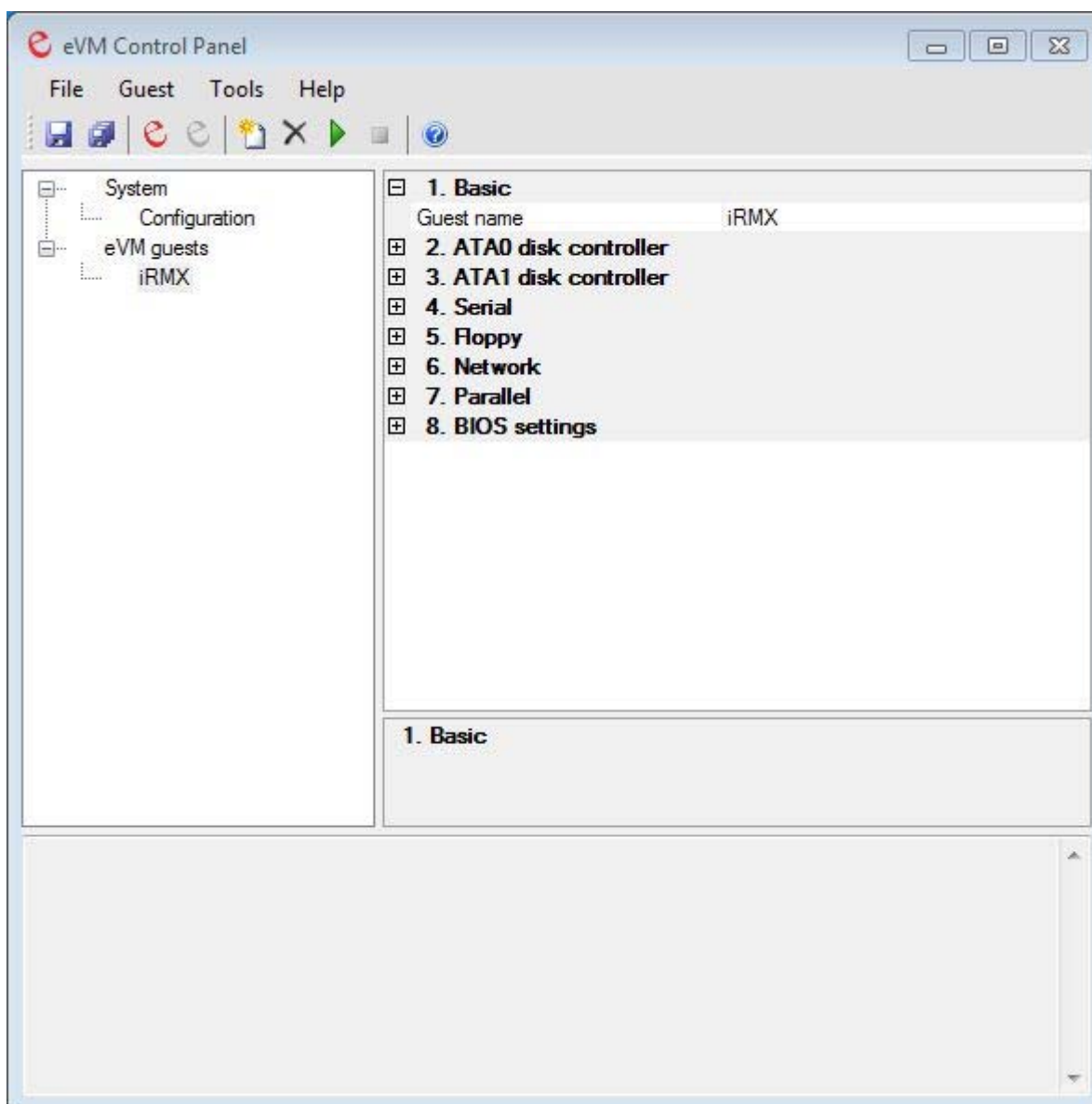


Figure 4.19 - Complete eVM Installation Showing iRMX Guest Configuration

17. Finally, let's look at the devices that are now in the Windows host system. Open Windows Device Manager by going to **Start->All Programs->Administrative Tools->Computer Management** and click on **Device Manager** Figure 4.20.

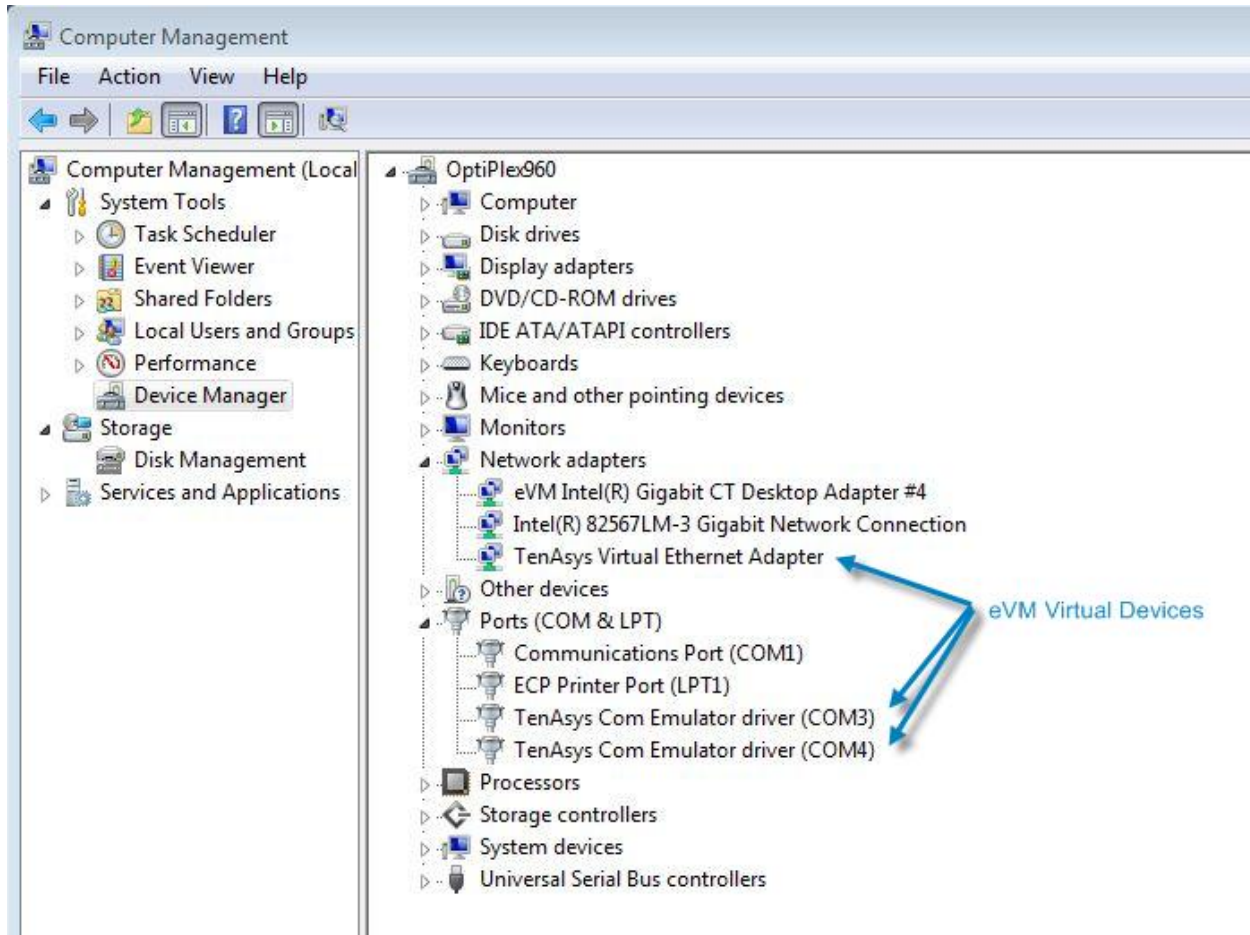


Figure 4.20 - Windows Device Manager

You can see that the TenAsys Virtual Ethernet Adapter has been installed, as well as, the two virtual serial ports. The property of each virtual serial port was to use the next available COM device, so you can see that in this system configuration, it resulted in the virtual serial ports becoming COM3 and COM4. We will discuss the configuration of the virtual Ethernet Adapter later.

4.2 Virtual Ports

Using eVM allows two discrete hardware platforms, each with its own operating system and applications, to be consolidated onto a single hardware platform. Not only does this provide a reduction in hardware required to host the two systems, it also provides a unique opportunity to simplify communications and connectivity between them. Instead of having real hardware communications devices on each discrete hardware platform that are hard-wired together, eVM

can provide a virtual software connection between the two systems, further reducing the hardware requirement to host the two interconnected systems.

4.2.1 Installation of Virtual Serial Ports

When eVM was installed, 2 virtual serial ports were also installed. If you want to add additional serial ports, eVM supports up to 4 virtual serial ports, you can add them through the eVM Control Panel. Let's look at a system with no virtual serial ports installed and look at how to install and configure a virtual serial port, a virtual COM device. Each virtual serial port provides a virtual COM device for the virtual machine that the guest OS can use for serial communications and a corresponding virtual COM device in Windows that will look like a standard COM device that is wired to the virtual COM device in the virtual machine. eVM manages this connection between the two COM devices using a shared memory interface.

1. Open the eVM Control Panel, and in the left panel under System, click **Configuration**, Figure 4.21.

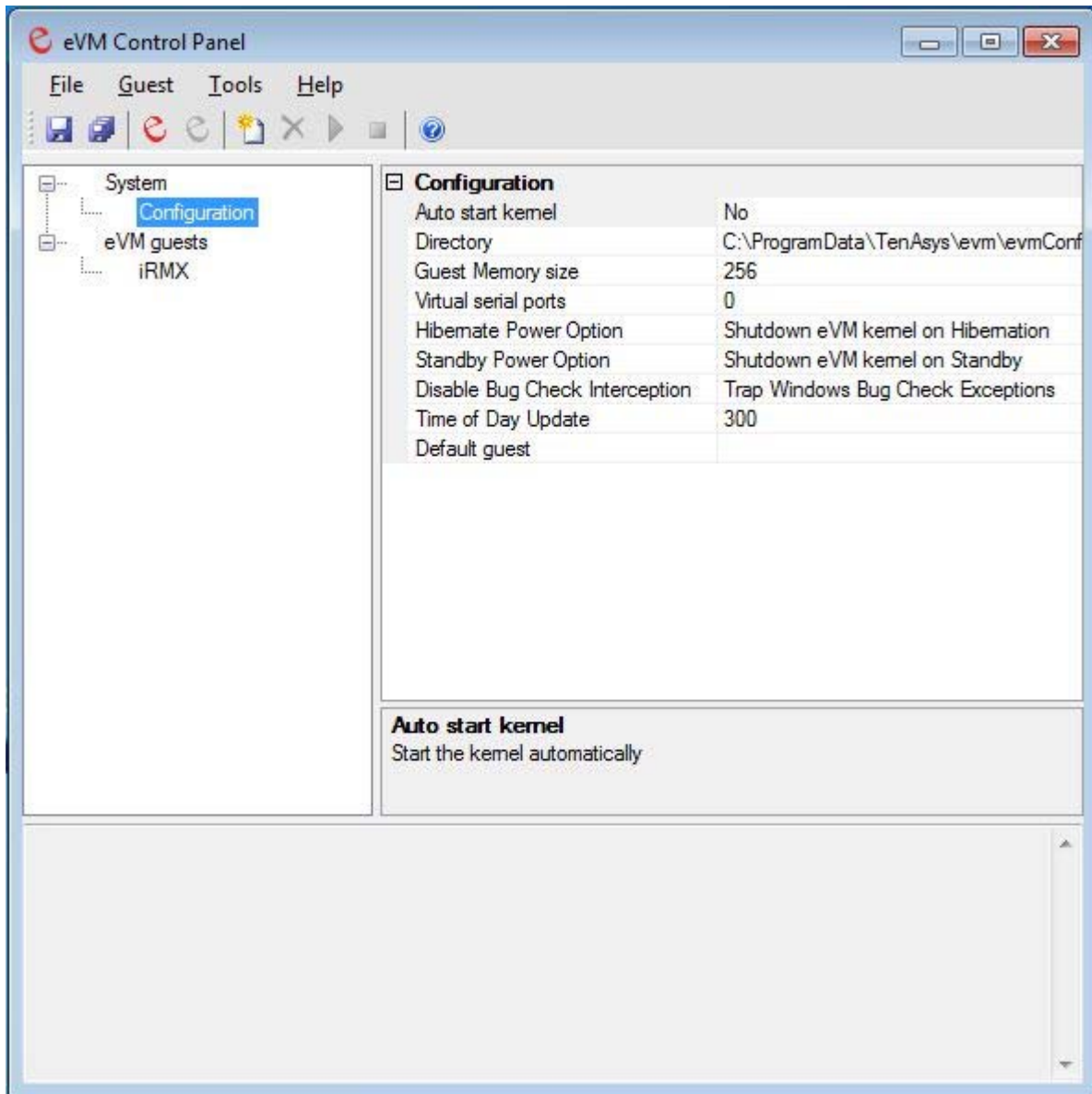


Figure 4.21 - eVM Control Panel Showing System Configuration with No Virtual Serial Ports

In the right pane of the eVM control Panel, the system configuration parameters are shown.

Note that the Virtual serial ports parameter is 0 for this example.

2. Double-click on the '0' value in the box next to *Virtual serial ports*, and enter '1' Figure 4.22

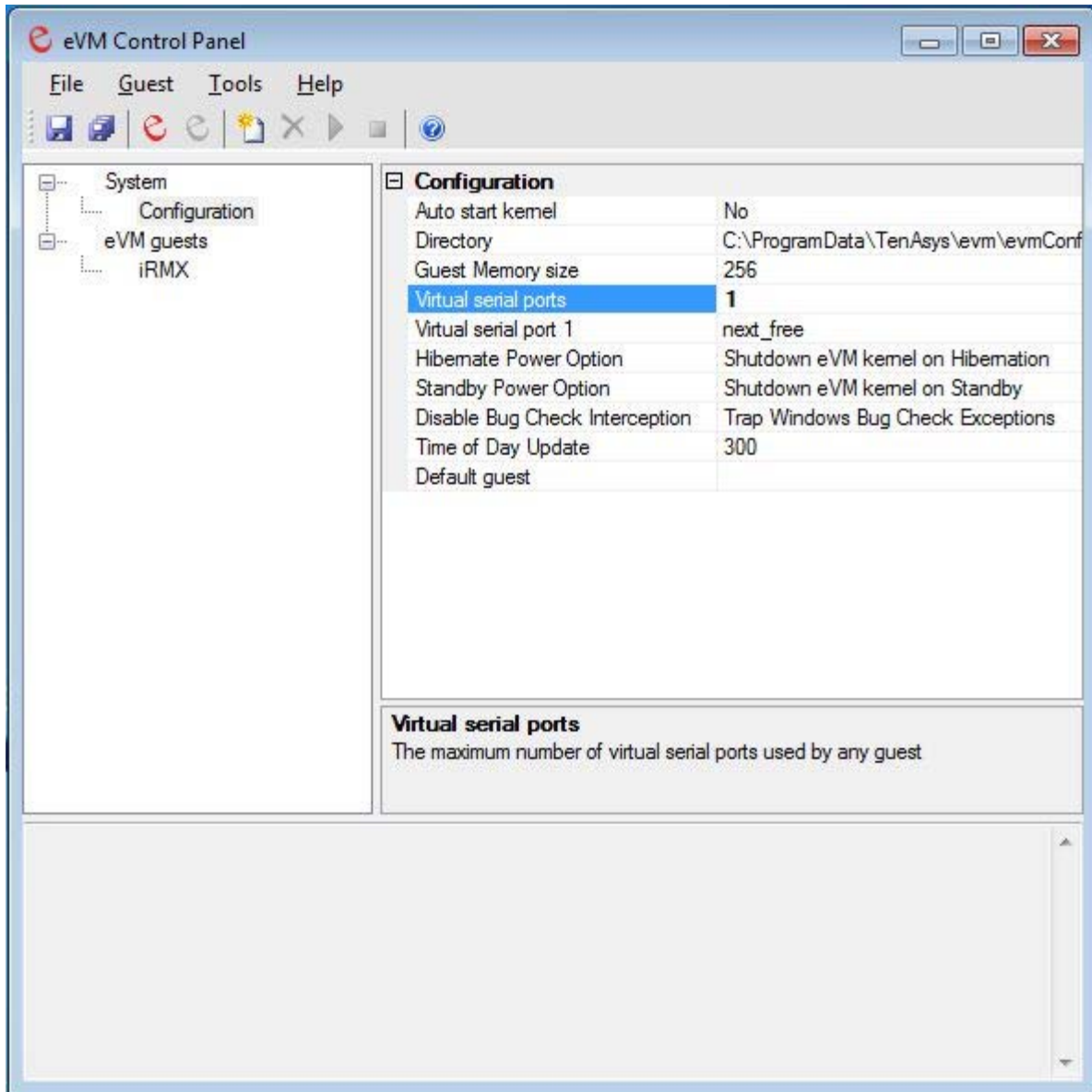


Figure 4.22 - eVM Control Panel, Virtual Serial Port Added

You will note that a new parameter has been added under *Virtual serial ports*, *Virtual serial port 1*. This is the new virtual COM device that has been added, and the default is to let eVM assign the COM port number to the next available. If a specific COM port number would be desired, it could be specified with the *Virtual serial port 1* parameter. If you click on the dropdown on the box next to *Virtual serial port 1*, you will see the COM port choices, Figure 4.23. We will leave the selection for *next_free* for this example.

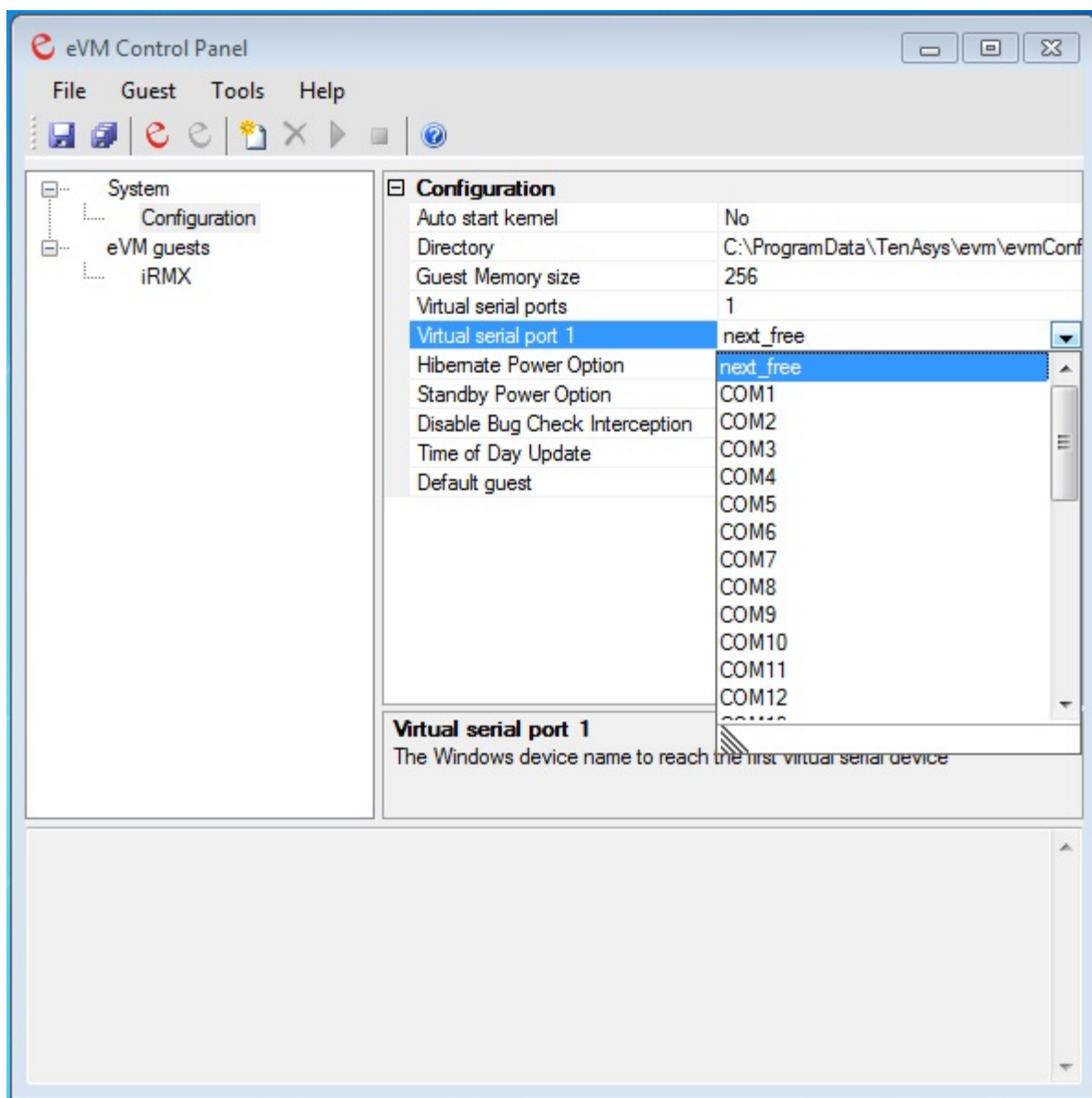


Figure 4.23 - Virtual Serial Port COM Port Selections

- To apply this configuration change, you must save the new configuration. From the eVM Control Panel dropdown, select **File->Save all** Figure 4.24.

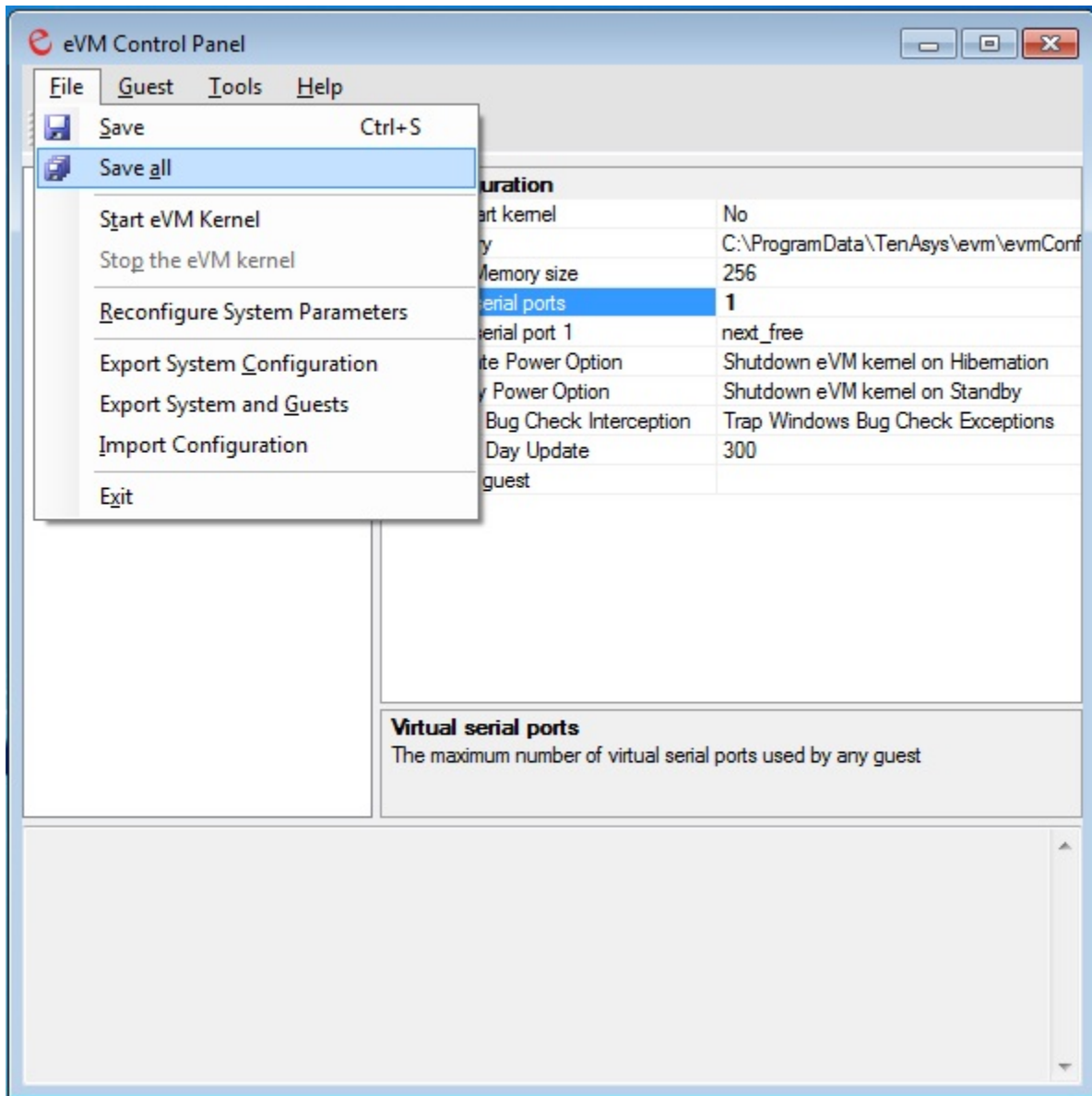


Figure 4.24 - Saving eVM Control Panel Configuration

4. Now, let's confirm that the virtual serial port has been added. Open Windows Device Manager by going to **Start->All Programs->Administrative Tools->Computer Management** and click on **Device Manager**. Under Ports (COM & LPT), you will now see the virtual serial port: *INtime Com Emulator driver (COM3)*, Figure 4.25.

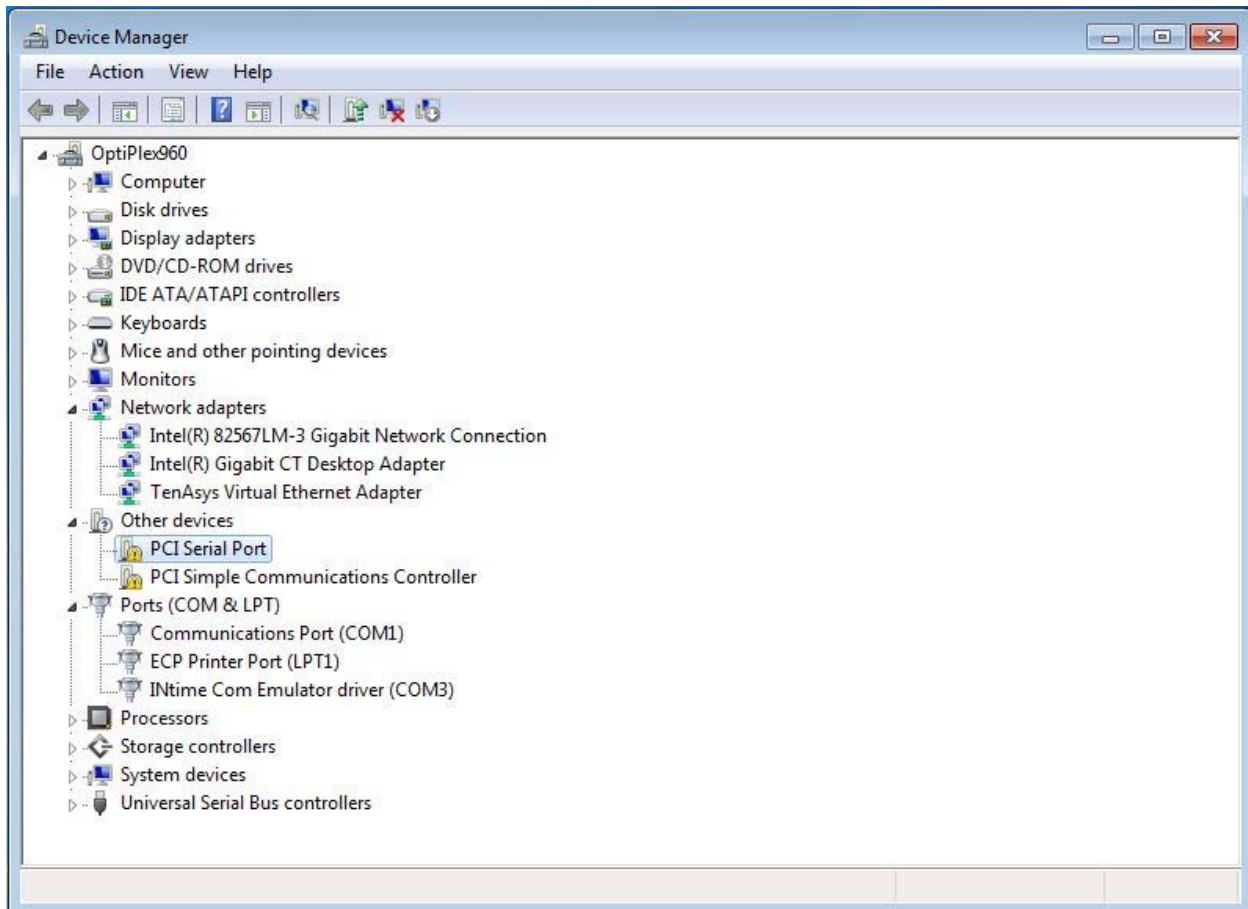


Figure 4.25 - Windows Device Manager Showing Virtual Serial Port

Applications in Windows can now access this virtual COM port by connecting to COM3.

5. Now, let's connect the virtual COM ports in the guest OS to the virtual COM ports of the Windows. Open eVM Control Panel, select the guest OS in the left pane (in this example it is iRMX), and expand the serial section in the right pane, Figure 4.26.

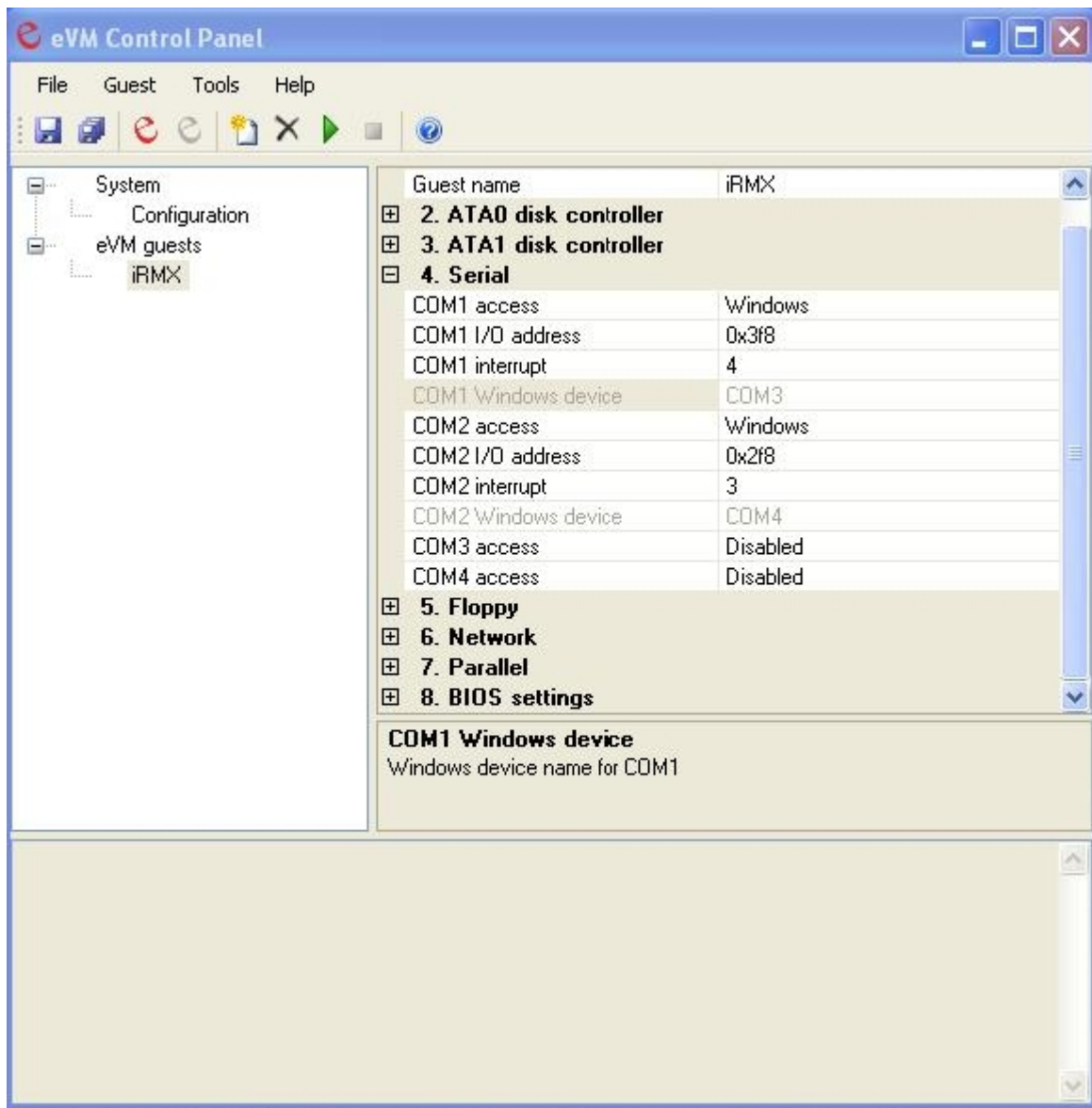


Figure 4.26 - eVM Control Panel Showing Guest Serial Device Settings

6. For each COM port that is to be connected to a virtual COM device in Windows, select the **COMn access**, and select **Windows**. In this example, iRMX has been set up for COM1 and COM1 to access Windows virtual COM devices.
7. Then to select the COMn device in Windows that the guest COMm device will connect to, select **COMn Windows device**, and select the desired COMm device for Windows. For this example iRMX COM1 will be connected to Windows COM3, and iRMX COM2 will be connected to Windows COM4.

4.2.2 Configuring the Virtual Ethernet Device

In this example we look at the case where we have two platforms interconnected via Ethernet that are now consolidated onto one platform, with the Ethernet connection being managed by eVM via virtual Ethernet devices. The consolidated system will consist of Windows as the host OS and Windows CE as the guest RTOS.

There are two parts to setting up eVM to manage the Virtual Ethernet connection, the virtual Ethernet device on the Windows side, and the virtual Ethernet device on the guest OS side.

Configuring the Windows side:

The follow step-by-step guide is for Windows 7/Vista. If you have installed eVM for Windows on Windows XP, following the step-by-step guide in Appendix G.

1. Open the **Windows Control Panel** and click on **Network and Sharing Center**, Figure 4.27.

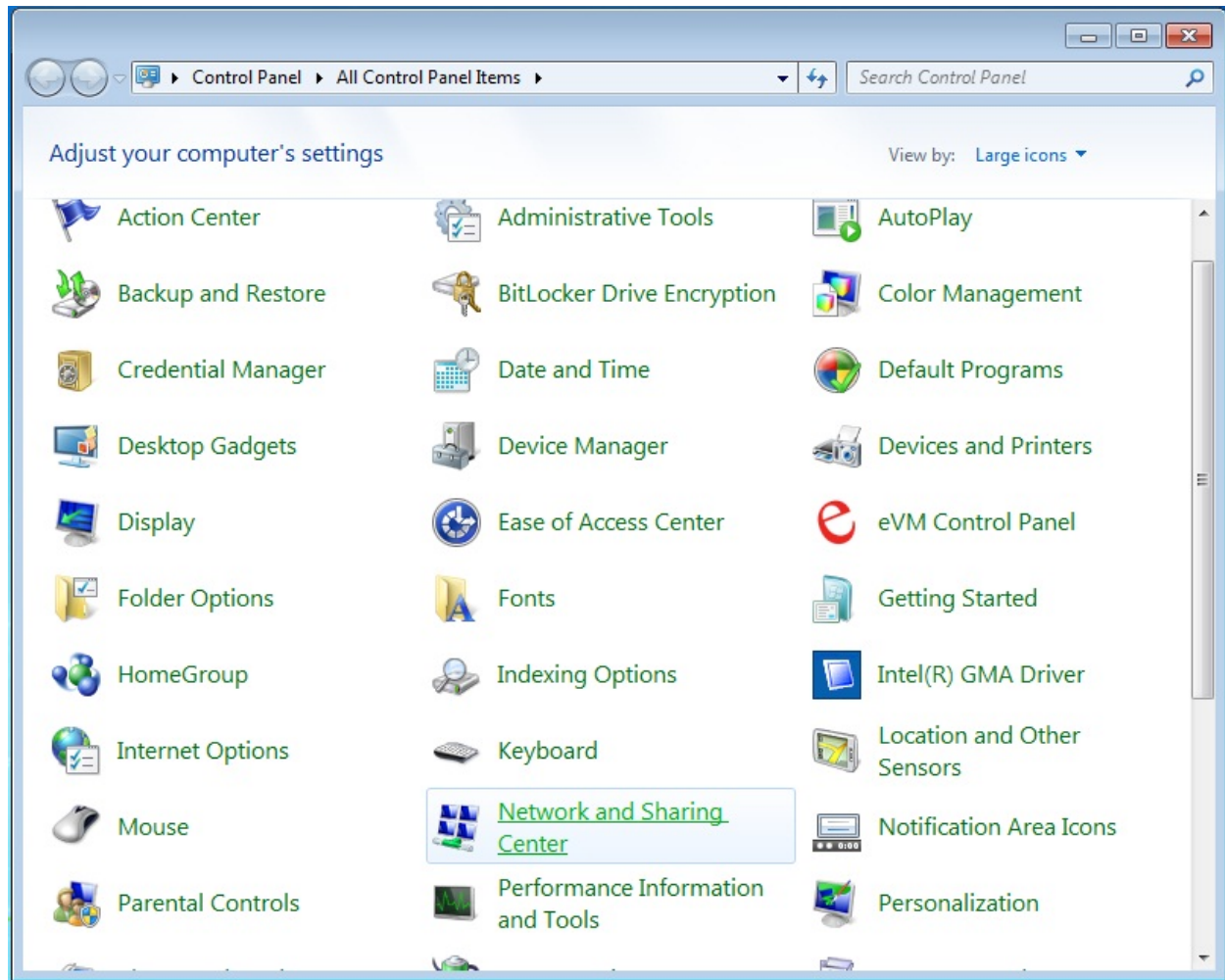


Figure 4.27 - Windows Control Panel, Select Network

2. In the upper left of the *Network and Sharing Center*, click on **Change adapter settings**,
Figure 4.28.

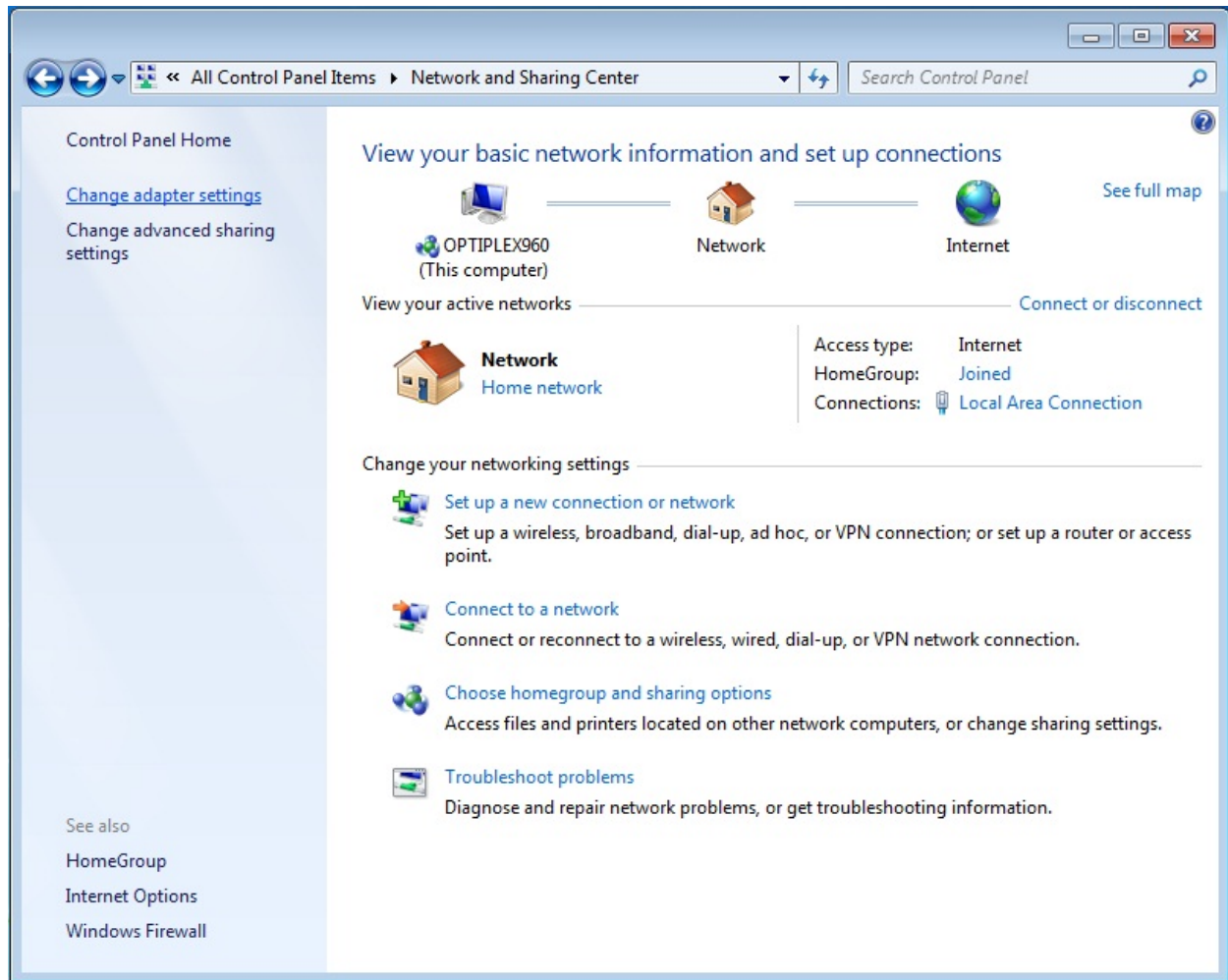


Figure 4.28 - Windows Network and Sharing Center

3. Now you will see *Network Connections*, and one of the network connections displayed is the *TenAsys Virtual Ethernet Adapter*, Figure 4.29. Double-click on the **TenAsys Virtual Ethernet Adapter**.

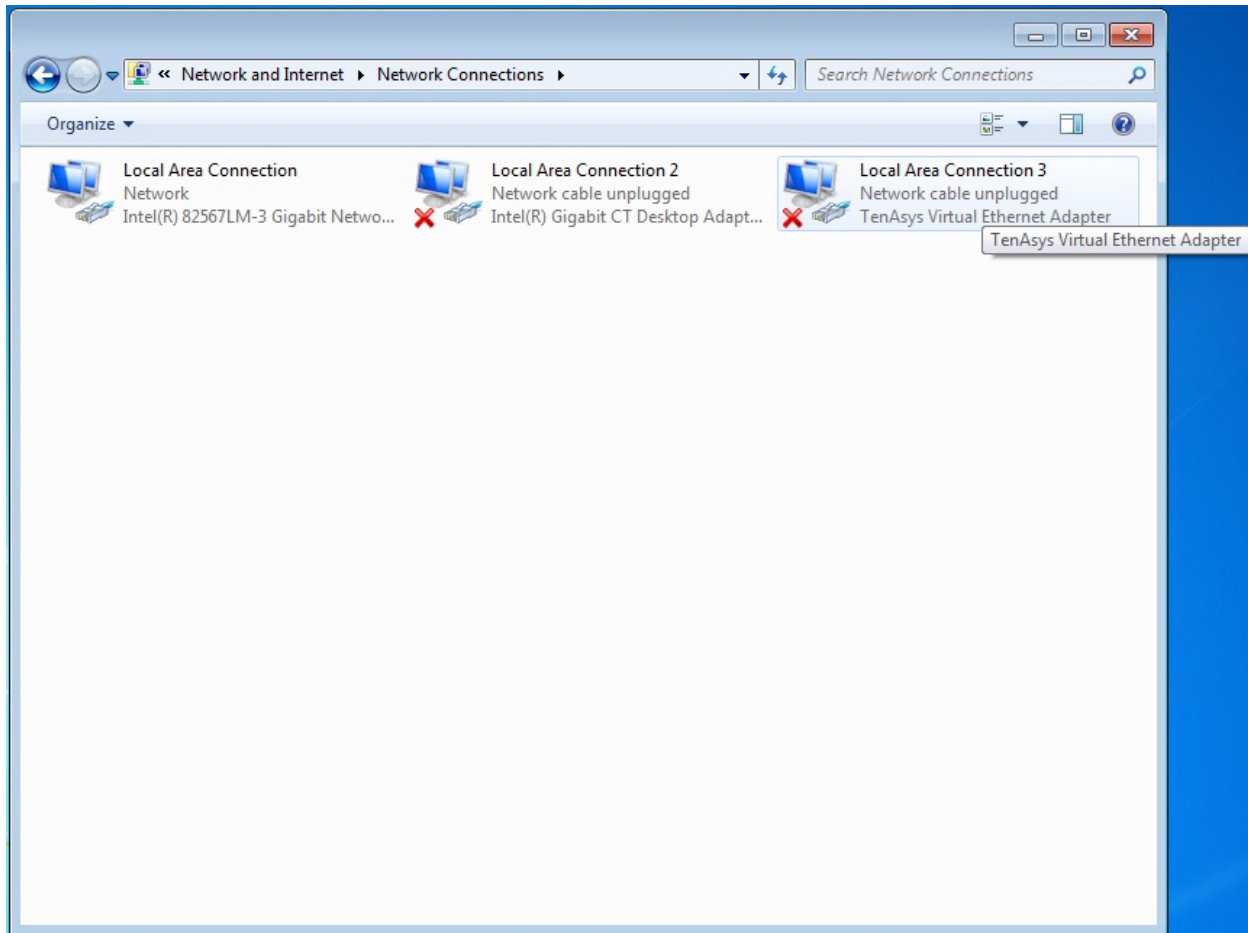


Figure 4.29 - Windows Network Connections

4. You will now see the *Local Area Connection Properties* for the *TenAsys virtual Ethernet Adapter*. We will be using TCP/IPv4, so uncheck the checkbox for **Internet Protocol 6 (TCP/IPv6)**, Figure 4.30.

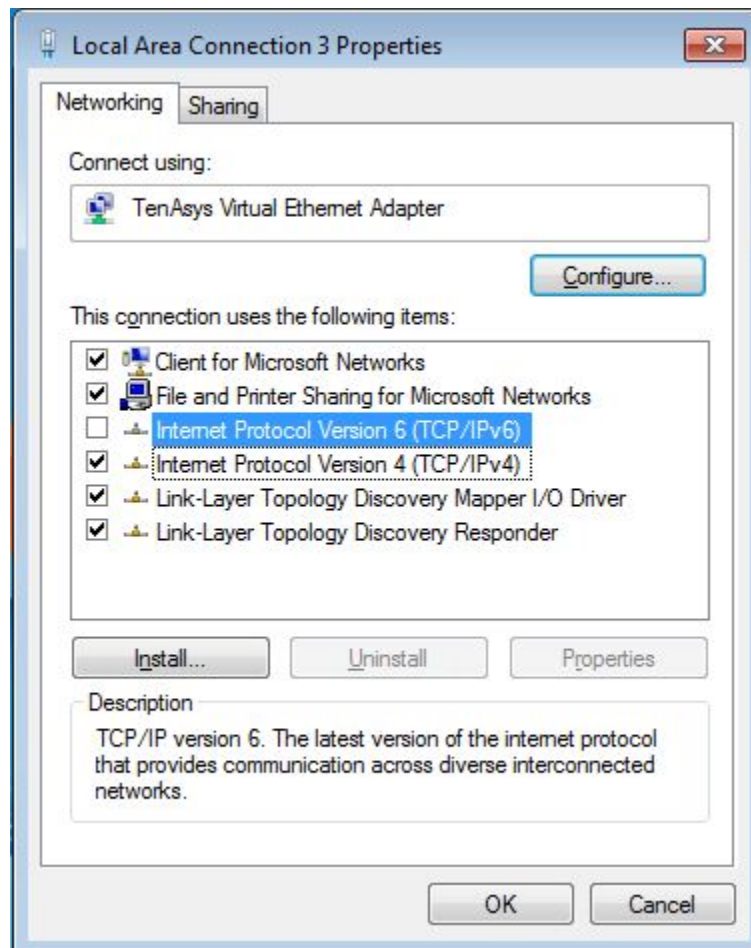


Figure 4.30 - TenAsys Virtual Ethernet Adapter, Windows Local Area Connection Properties

5. Select Internet Protocol Version 4(TCP/IPv4), and click the Properties button, Figure 4.31.

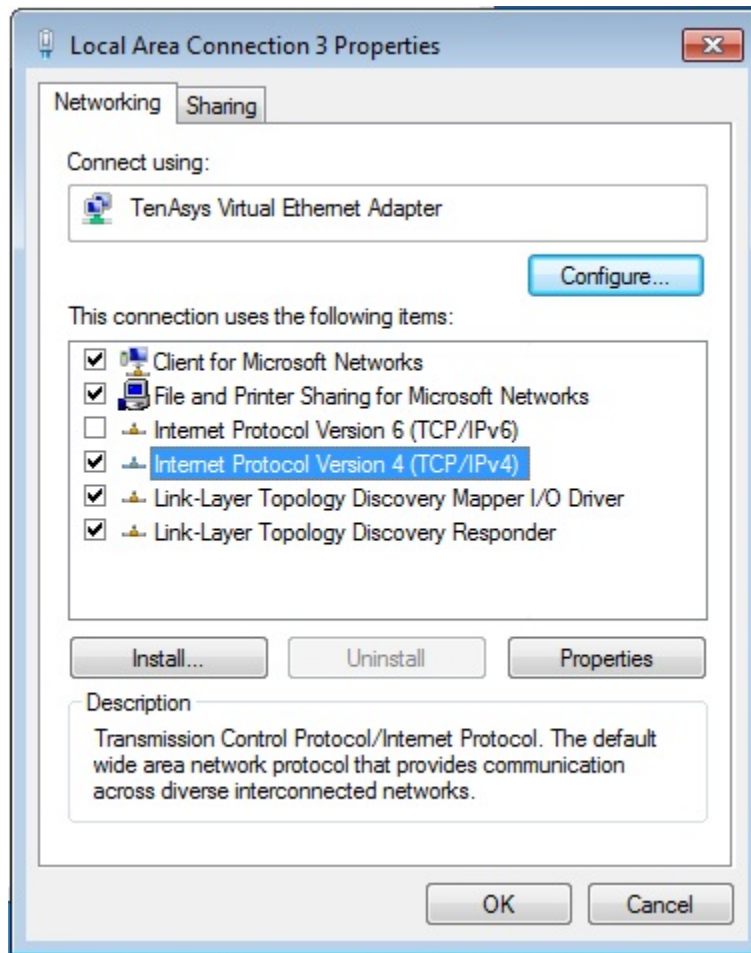


Figure 4.31 - Selecting TenAsys Virtual Ethernet Adapter TCP/IPv4 Properties

6. In the *Internet Protocol Version 4 (TCP/IPv4) Properties* window, select the **Use the following IP address:** radio button, and enter **10.0.0.1** in the IP address. Keep the default Subnet mask: of **255.0.0.0**, Figure 4.32.

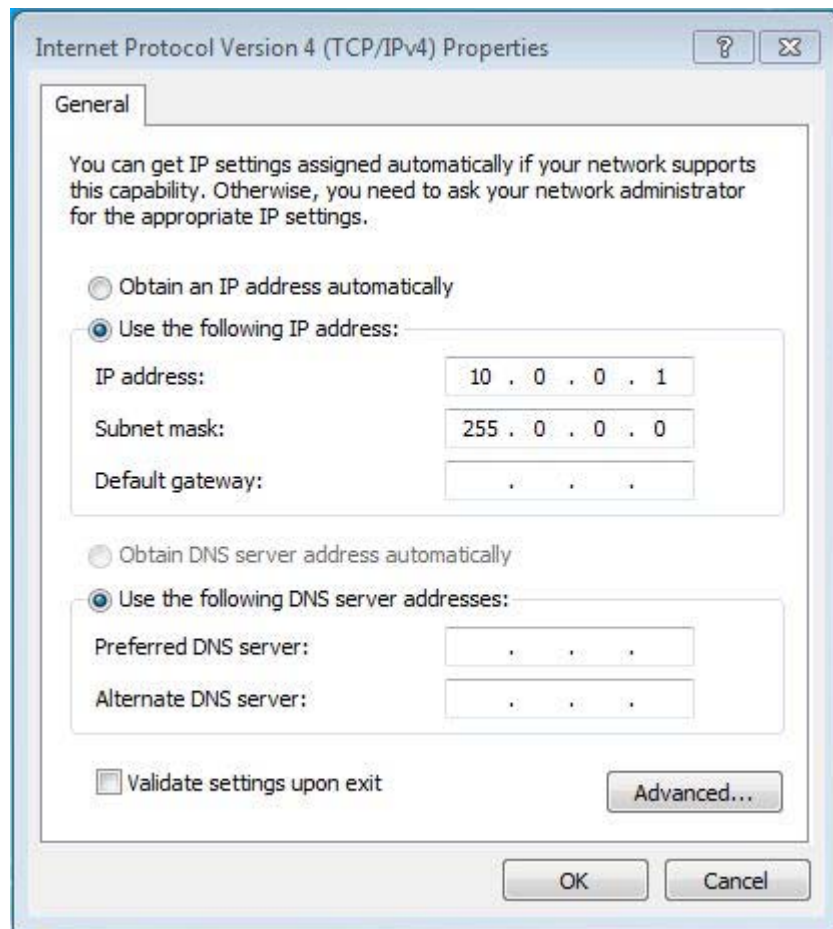


Figure 4.32 - Setting the Static IP Address for the TenAsys Virtual Ethernet Adapter

7. To accept this new IP address setting, click **OK**.
8. Close the *Network Connections* window, and the Windows side of the virtual Ethernet connection configuration is complete.

Configuring the guest OS side:

1. Open **eVM Control Panel**, select the **guest OS**, and select the **Network** section, which has an item named *Virtual Ethernet (NE2000)*, Figure 4.33.

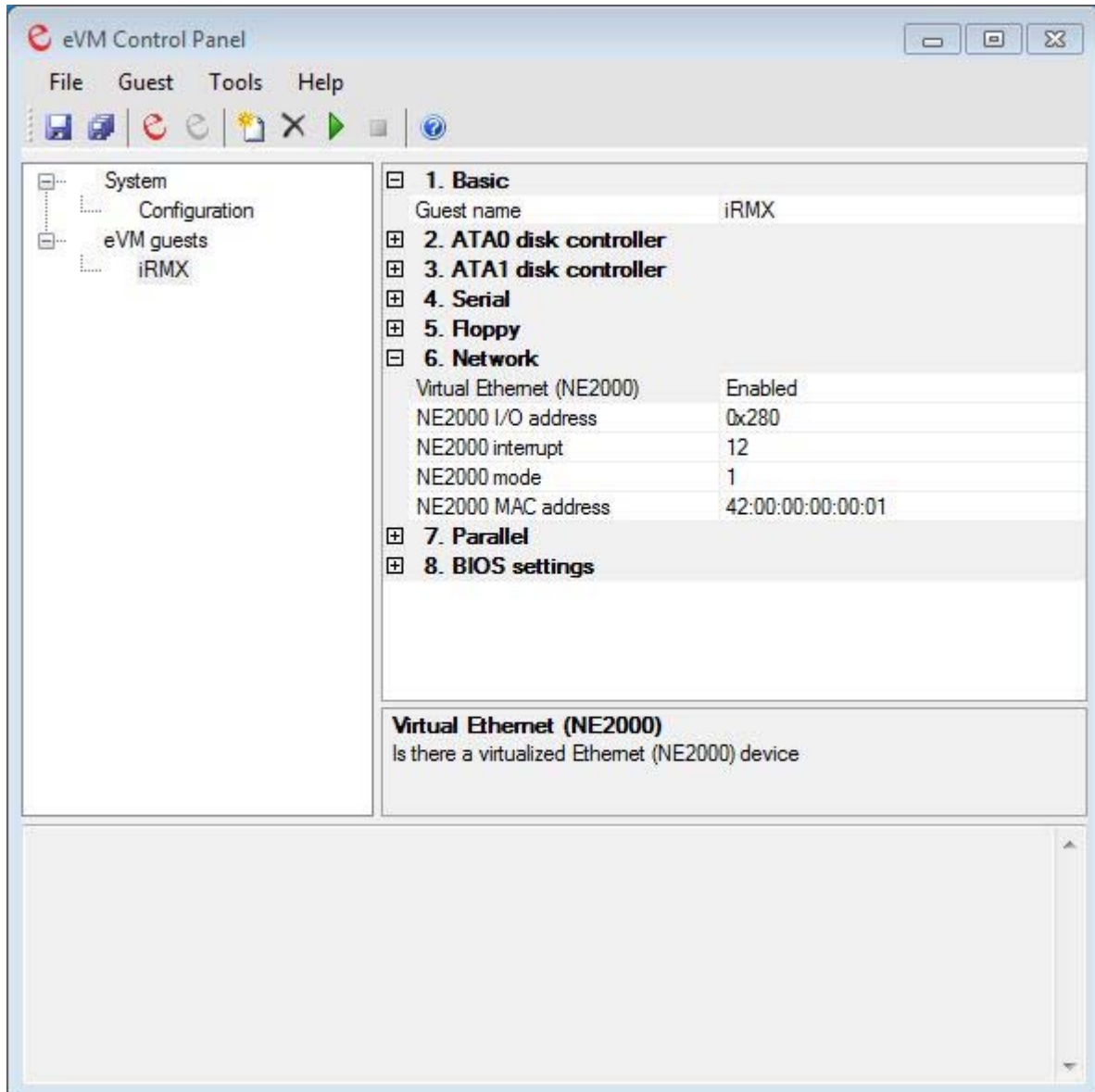


Figure 4.33 - eVM Control Panel, Guest OS Virtual Ethernet Device Configuration

1. In the box next to *Virtual Ethernet (NE2000)*, select **Enabled** to enable the virtual Ethernet device.
2. You will now see the configuration for the virtual Ethernet adapter for the virtual machine. Match the I/O address, and interrupt to what the Guest OS expects.

Note: we will discuss more about the guest OS device parameters in general, and with respect to QNX and Windows CE specifically in Section 4.5.

3. The guest OS needs to be configured to use an IP address on the same sub-net as the virtual Ethernet adapter on the Windows side. In this example with iRMX, we entered 10.0.0.2 as the IP address in the boot loader. The guest OS now sees a virtual NE2000 Ethernet device at I/O 0x280, interrupt 12 and configures its NE2000 driver to use IP address 10.0.0.2 for that device.

The actual guest OS needs to talk to the IP address of the Windows side (10.0.0.1); and when the Windows side responds, it needs to know to talk to the IP address of the guest OS side (10.0.0.2). When we did our Windows CE example, we told the Windows host to find a client at 10.0.0.2 and send the image to it.

4.3 Running the Sample Guest RTOS, iRMX

Now that we have eVM installed and a virtual COM device installed and configured, let's run the sample guest RTOS, iRMX.

1. In Windows, open a terminal emulator program, and configure it to connect to the first virtual COM device. Recall from Figure 4.20 that the first virtual COM device was COM3 in our example.

2. Open **eVM Control Panel** and select **iRMX** under eVM guests, Figure 4.34.

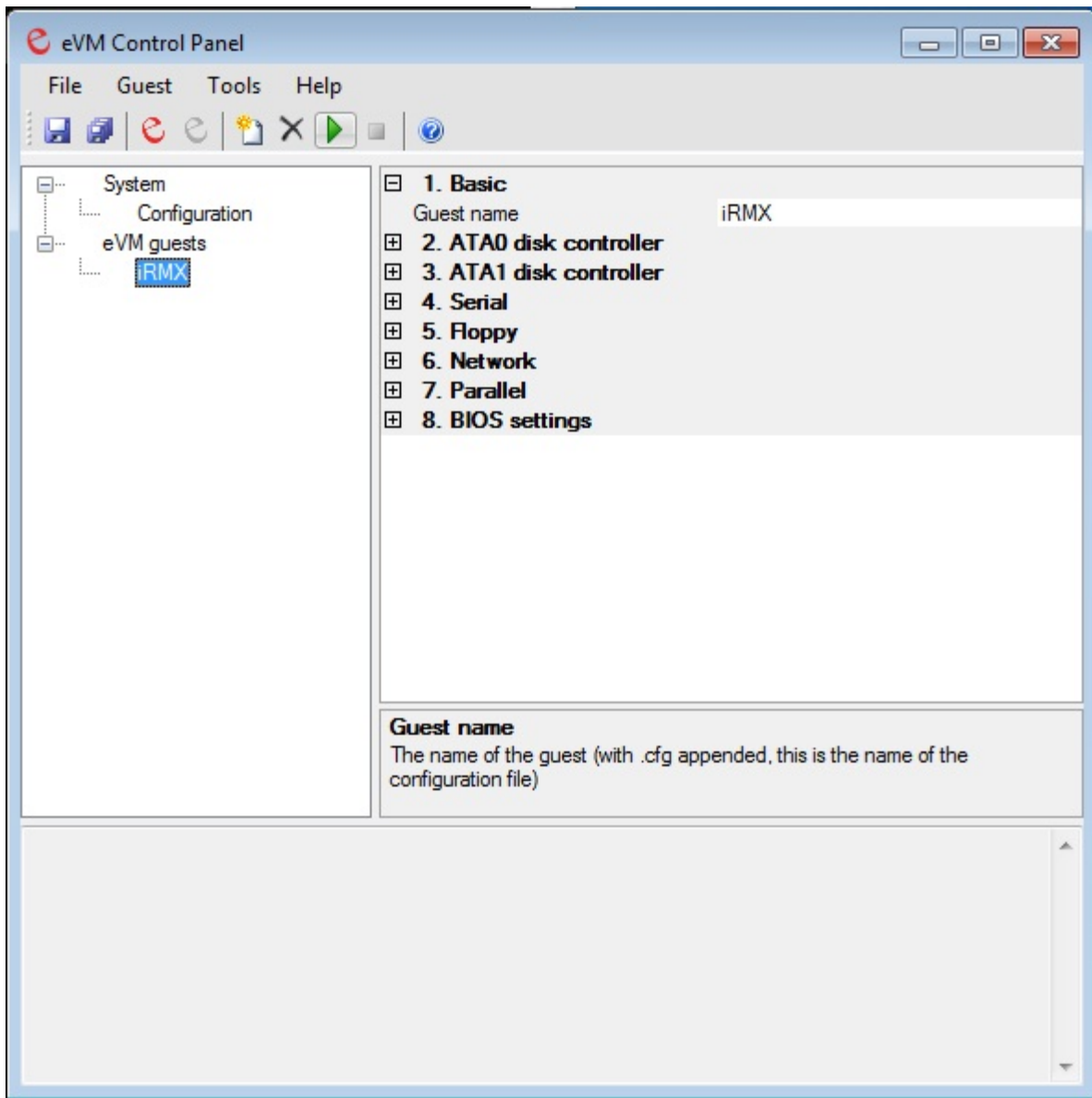


Figure 4.34 - eVM Control Panel, Selecting the Guest OS

3. From the toolbar, click the **start** button (green triangle) Figure 4.35.

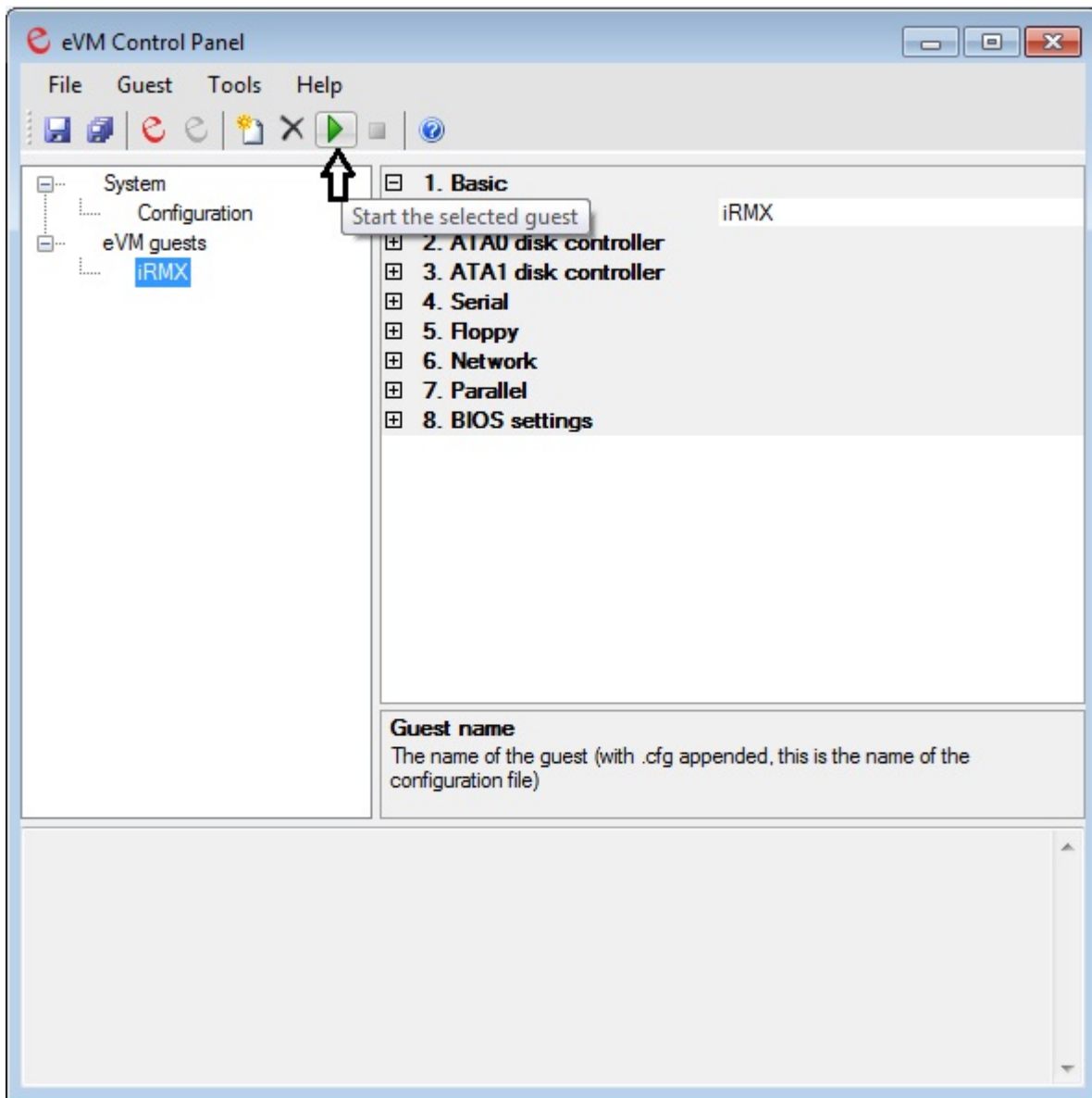


Figure 4.35 - eVM Control Panel, Start the Selected Guest OS

4. The guest OS, iRMX, will boot in the virtual machine, an eVM Console window will open, and the default debug output from iRMX will be output to the virtual COM device and show up in the terminal emulator window, Figure 4.36.

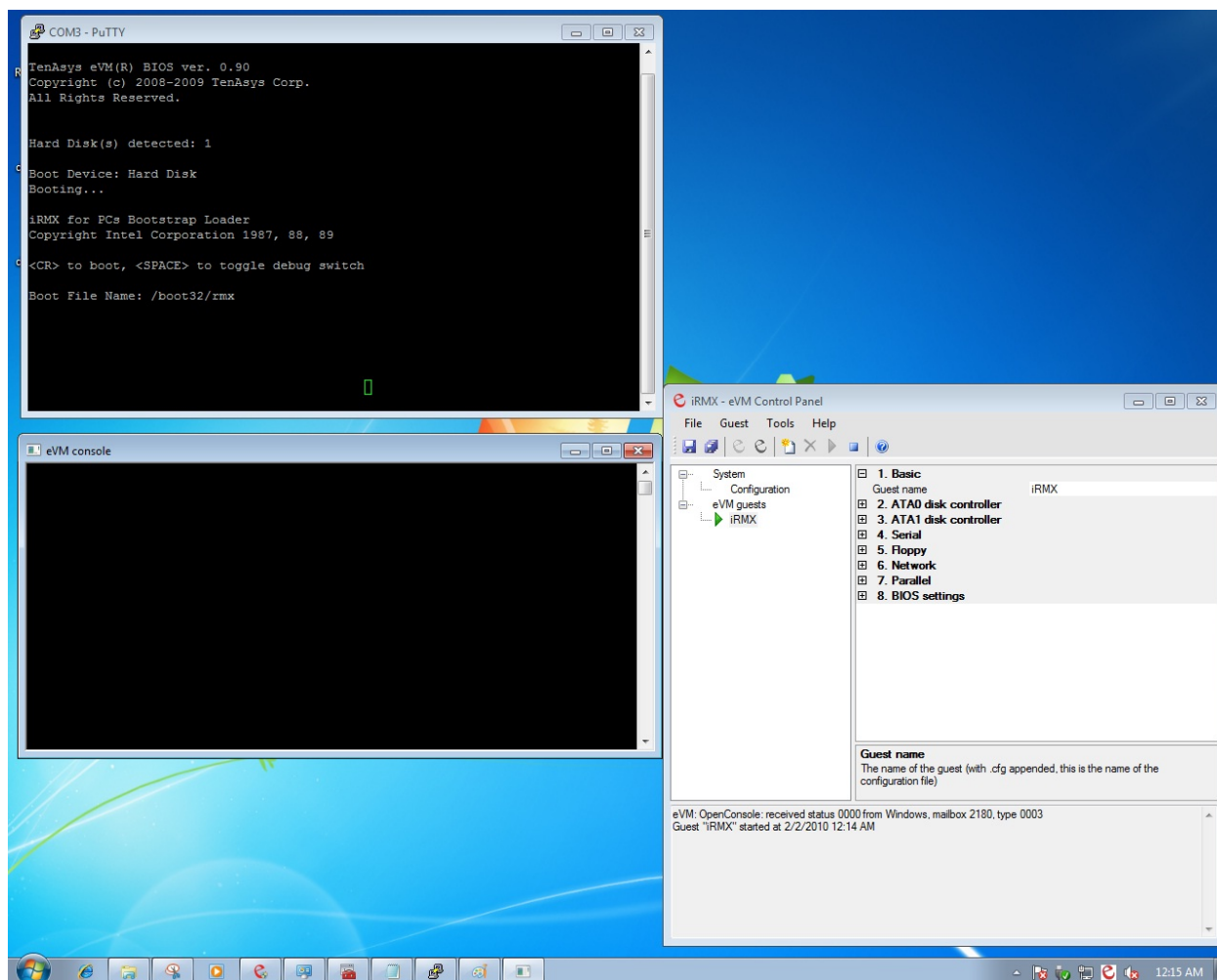


Figure 4.36 - Guest RTOS, iRMX, Booting in the Virtual Machine

Note: for this example we are using the freeware PuTTY application, but any terminal emulation application can be used.

5. Once iRMX boots, you can now type iRMX system commands in the terminal emulator and iRMX will respond exactly as if it were running on its own dedicated hardware platform.
6. To stop iRMX, in the eVM Control Panel from the toolbar, select the **stop** button (blue square); and the eVM Control Panel will prompt you if you want to stop the guest OS.

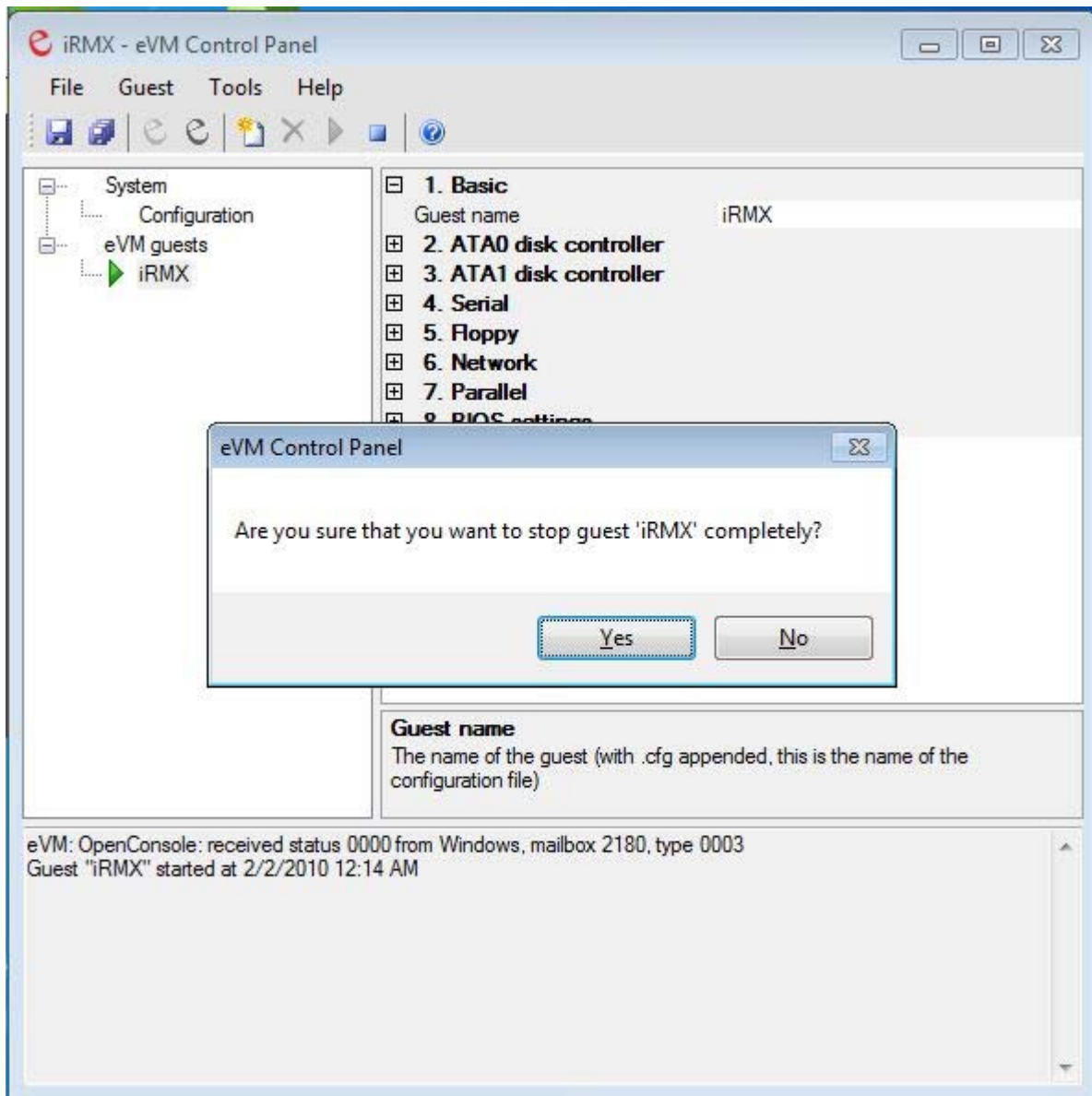


Figure 4.37 - eVM Control Panel, Stopping the Guest OS

7. Click **Yes**, and iRMX will be stopped.

4.4 Installation of a Guest OS

Installing a guest OS on eVM consists of obtaining a disk or image of the target system, and creating a configuration for the guest OS.

The disk image can either be copied from an existing system; or in some cases, it can be installed directly from the installation materials. The configuration is created through the eVM Control Panel utility.

For a list of enabled guest OSes, see Appendix C.

4.4.1 Configure your existing system

By reference to the Virtual Machine specification, Appendix B.2, configure your system in order to match the Virtual Machine hardware interface. The primary point to notice is the absence of a video device and keyboard, so any console usually has to be directed to a serial port by configuration.

4.4.2 Make a disk image of your system

When the system is configured, make a disk image by using a suitable tool. In some cases the disk could be attached to your Windows system and an imaging tool used to make a binary image of the disk. If you are not going to copy the entire drive, make sure you have included the Master Boot Record (MBR) of the disk, as well as, the partition containing your software. There are many Windows tools available to do this, such as the shareware utility, SelfImage.

If you are making an image from a Compact Flash (CF) drive, you may need to adjust the CHS values in the configuration file so that it will boot correctly. Some CF parts have unusual configurations, and these are typically not copied with the data from the drive.

When you have a disk image, copy it to the directory where eVM stores its images. This varies depending on which version of Windows you are using, but the environment variable %EVMCFG% always contains the name of this directory. There is also a link to it in the Start menu. Go to Start->eVM for Windows->eVM Data Folder to open an Explorer Window in this location. Copy your disk image file here.

4.4.3 Make a guest configuration

Now you have a disk image, you need to create a configuration for your system. We will assume that your system has one disk image, and its console is the first serial port (COM1).

1. Open the eVM Control Panel (use the eVM system tray icon or navigate to **Control Panel->eVM Control Panel**).
2. Start a new configuration by clicking on **Guest->New**.
3. In section 1 of the right-hand panel, "Basic", name your configuration. Configuration names may have up to 12 characters.
4. Assign your disk image. In section 2, change "ATA0 enabled" to **Yes**.
5. In ATA0 master path, click the ... button at the right and navigate to your disk image and open it.

6. In section 4, "Serial", change "COM1 access" to "**Windows**". Note the Windows COM driver name. This is the other end of the virtual serial link from the guest to Windows, and is the device which your terminal emulator will open.
7. In section 8, "BIOS settings", change "Boot device 1" to **Disk** so the BIOS knows to look for a disk device to boot from.
8. Save your settings (**File->Save**).

You are now ready to test your configuration. Open a terminal emulator and use it to open the serial port you noted in step 6 above.

Start your guest by clicking on the **Start** button on the toolbar, or by selecting **Guest->Start** from the dropdown.

4.4.4 Installing your operating system from installation materials

In some cases you can install your target operating system using standard installation materials, so long as these do not depend on a video/keyboard console. If you have a floppy boot disk image you should copy this to the eVM Data Folder (%EVMCFG%) and assign the image to the floppy drive of your guest configuration. To do this:

1. Open eVM Control Panel and in the right panel, expand the Floppy device.
2. For *Floppy A enabled* select **Enabled**.
3. In **Floppy A path**, click on the ... button to the right, navigate to the floppy boot disk image file, and select it.
4. From the dropdown, select **File->Save all** to apply the change.

4.5 Specific steps for certain Guest OSes.

Two example configurations are illustrated here. For more examples, visit the TenAsys website at <http://www.tenasys.com/support>.

4.5.1 Installation and configuration of QNX 6.4

This is an example of customizing a target operating system and creating a disk image.

QNX 6.4 may be installed onto eVM for Windows in the following way: You will need to create a configuration which will boot without the use of the video card. A sample build file is included in this example:

1. Use mkifs to create a boot image from this build file, and then copy the boot image to /.boot.

```
#
# The build file for QNX Neutrino booting on a PC
#
[linker="ntox86-ld -T$QNX_TARGET/x86/lib/nto.link %(h!=0, -Ttext
0x%t%)(d!=0, -Tdata 0x%d%) -o%o %i %[M -L%^i -uinit_%n -lmod_%n%]"]
```

```
[virtual=x86,bios +compress] boot = {
    # Reserve 64k of video memory to handle multiple video cards
    startup-bios -s64k

    # PATH is the *safe* path for executables (confstr(_CS_PATH...))
    # LD_LIBRARY_PATH is the *safe* path for libraries
    (confstr(_CS_LIBPATH))
    # i.e. This is the path searched for libs in setuid/setgid
    executables.
    PATH=/proc/boot:/bin:/usr/bin:/opt/bin
    LD_LIBRARY_PATH=/proc/boot:/lib:/usr/lib:/lib/dll:/opt/lib procnto-instr
}

[+script] startup-script = {
    display_msg " "
    display_msg "QNX Neutrino 6.4 for eVM"
    display_msg " "

    # To save memory make everyone use the libc in the boot image!
    # For speed (less symbolic lookups) we point to libc.so.3 instead of
    libc.so
    procmgr_symlink ../../proc/boot/libc.so.3 /usr/lib/ldqnx.so.2

    display_msg "----> Starting PCI services"
    seedres
    pci-bios
    waitfor /dev/pci

    display_msg "----> Starting console service"
    devc-ser8250 -e -b57600 &
    waitfor /dev/ser1
    reopen /dev/ser1

    # Default user programs to priority 10, other scheduler (pri=100)
    # Tell "diskboot" this is a hard disk boot (-b1)
    # Tell "diskboot" to use DMA on IDE drives (-D1)
    # Start 4 text consoles by passing "-n4" to "devc-con" (-o)
    # By adding "-e" linux ext2 filesystem will be mounted as well.
    #[pri=100] PATH=/proc/boot diskboot -b1 -D0 -odevc-con,-n4 -odevc-con-
hid,-n4
    display_msg "----> Starting EIDE driver"
    devb-eide blk auto=partition dos exe=all cam quiet eide nobmstr
    waitfor /dev/hd0

    mount /dev/hd0t79 /
    mount -tcd /dev/cd0 /fs/cd0

    display_msg "----> Starting /dev/system/sysinit"
    ksh -c /etc/system/sysinit
```

```

}

[type=link] /dev/console=/dev/ser1

# Include the current "libc.so". It will be created as a real file using
# its internal "SONAME", with "libc.so" being a symlink to it. The symlink
# will point to the last "libc.so.*" so if an earlier libc is needed
# (e.g. libc.so.1) add it before the this line.
libc.so
libhiddi.so
libusbdi.so

# Include all the files for the default filesystems
libcam.so
io-blk.so
cam-disk.so
fs-qnx4.so
fs-qnx6.so
fs-dos.so
fs-ext2.so
cam-cdrom.so
fs-cd.so
fs-udf.so

# USB for console driver
devu-ehci.so
devu-ohci.so
devu-uhci.so
devh-usb.so
devh-ps2ser.so

[data=copy]
seedres
pci-bios
devb-eide
devb-umass
devb-ahci
devb-mvSata
devb-adpu320
devb-aha8
umass-enum
[search=${MKIFS_PATH}:${QNX_TARGET}/etc] umass-enum.cfg
io-usb
io-hid
diskboot
slogger
fesh
mount
devc-ser8250

```

```
#devc-con
#devc-con-hid
```

```
# These files will be unlinked after the system has started
# to release the memory back to the system. They are only
# needed during boot. If needed later on, they will be loaded
# from the boot device.
unlink_list={
/proc/boot/devb-*
}
```

2. Now, configure your console device by editing **/etc/config/ttys** and add a line for **/dev/ser1**:

```
ser1 "/bin/login" qansi-m on
```

3. Configure the network parameters for the virtual Ethernet subnet between Windows and QNX. We are going to define a subnet of 10.100.200.0, and give it two nodes, one for Windows and one for QNX. Edit **rc.local** to load the **ne2000 device** and configure it by adding the following lines:

```
io-pkt-v4 -dne2000 irq=12,ioport=0x280 -ptcpip
ifconfig en0 10.100.200.2 up
inetd &
qconn &
```

4. In Windows, edit the properties of the TenAsys Virtual Ethernet network connection and set the IP address to **10.100.200.1**.
5. Now, make an image of your disk. You can use **dd** and write the image to another partition, or device (for example):

```
dd if=/dev/hd0 of=/mnt/tmp/qnximg.bin count=9900
```

Note: the size (ifs) parameter must include both the boot sector and the partition that you want to include in the image.

6. Copy the output file to your Windows %EVMCFG% directory and assign it as the first hard drive for your QNX configuration.
7. Make sure that you have at least one virtual serial terminal defined and add it to the configuration.

4.5.2 Installation and configuration of Windows CE

This is an example of installing a guest operating system using the materials provided by the vendor.

1. Create a DOS-bootable floppy diskette.
2. Copy the following files to it from your Windows CE 6.0 installation:
loadcepc.exe
eboot.bin
3. Create an autoexec.bat file with these commands in it:

```
@echo off  
verify off  
loadcepc /v /e:280:12:192.168.99.2 EB00T.BIN
```

Note: this will load KITL using the NE2000 virtual Ethernet driver.

4. Configure your Windows Virtual Ethernet driver to use IP address **192.168.99.1**.
5. Now, make a disk image of your floppy diskette, and copy it to the %EVMCFG% directory.
6. Add a floppy device to your configuration and assign this file to it.

Note: this will load the KITL components, so you can now connect to the target using your Windows CE 6.0 installation.

4.6 Driver configuration and setup

Now that we have eVM installed under Windows, the host OS, and have installed a guest OS, we need to dedicate devices to the host OS, to the guest OS, or share them between both. This section will describe how to use eVM Device Manager to accomplish this.

4.6.1 eVM Device Manager

The eVM Device Manager can be launched from the eVM Control Panel by selecting **Tools->Device Management** from the eVM Control Panel dropdown, Figure 4.38, in Windows Vista or Windows 7 by clicking the **Windows Start button** and entering "**devconfig.exe**" into *Search programs and files* and click on **devconfig.exe**, Figure 4.39, or in Windows XP by selecting **Start->Run**, enter "**devconfig.exe**", and click **OK**, Figure 4.40.

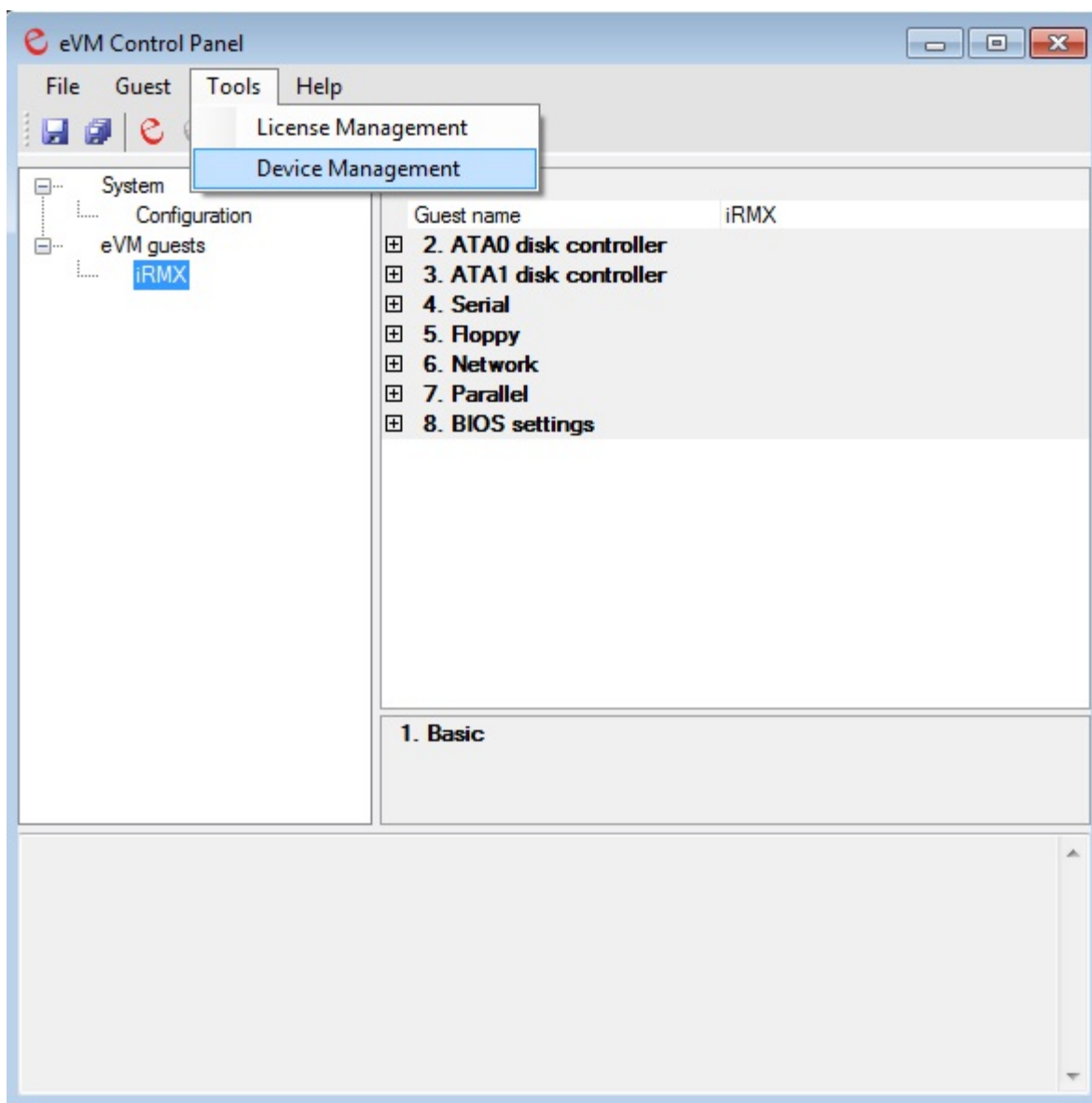


Figure 4.38 - Launching eVM Device Manager from eVM Control Panel

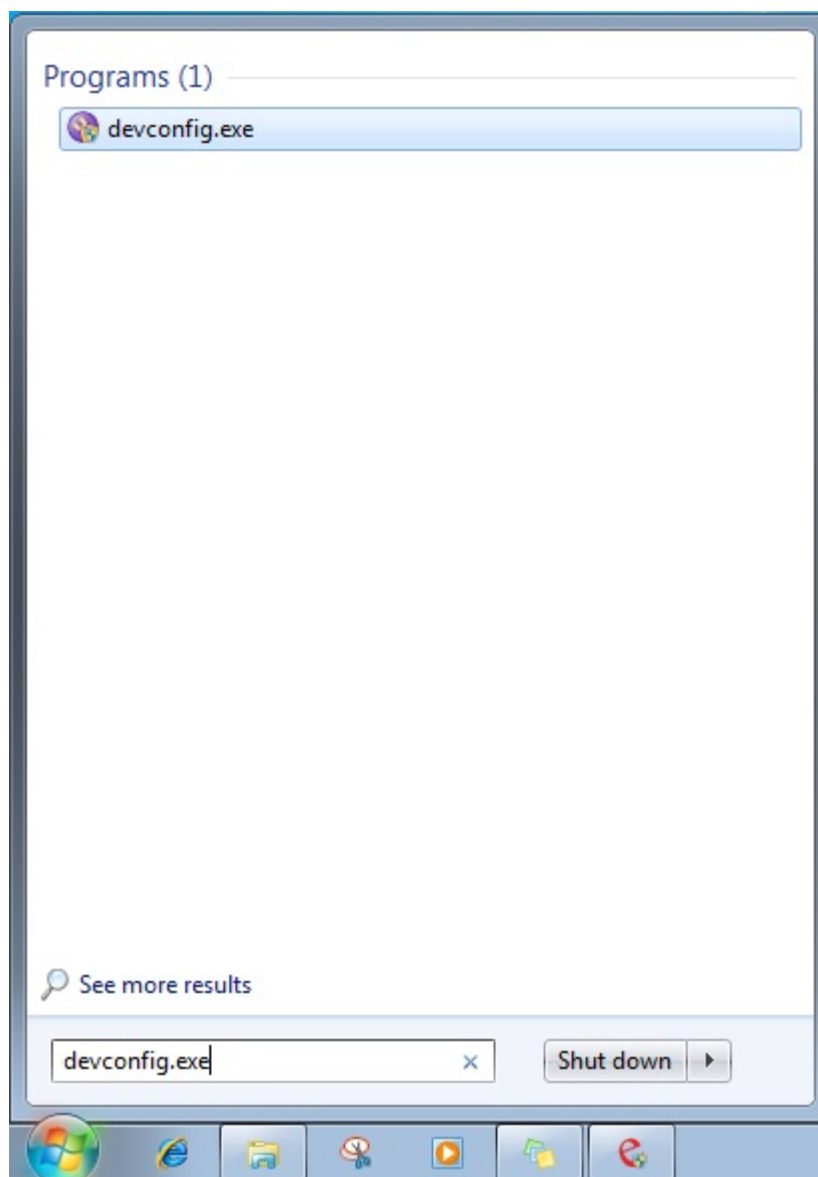


Figure 4.39 - eVM Device Manager launch from Windows Vista or Windows 7

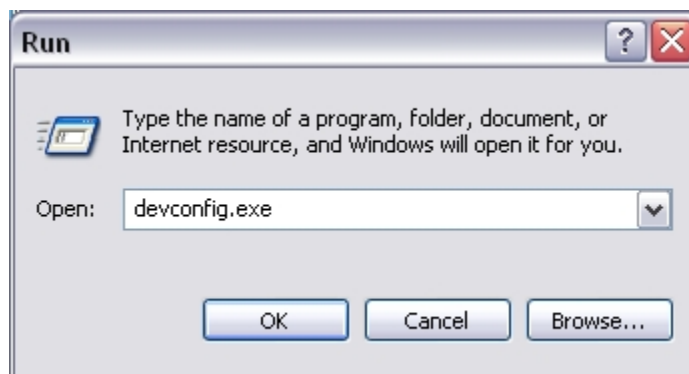


Figure 4.40 - eVM Device Manager launch from Windows XP

When the eVM Device Manager launches, you will see two active panes, Figure 4.41. The left pane shows two trees of device classes and devices: one controlled by Windows and the other managed by eVM. In either tree you can expand and collapse classes.

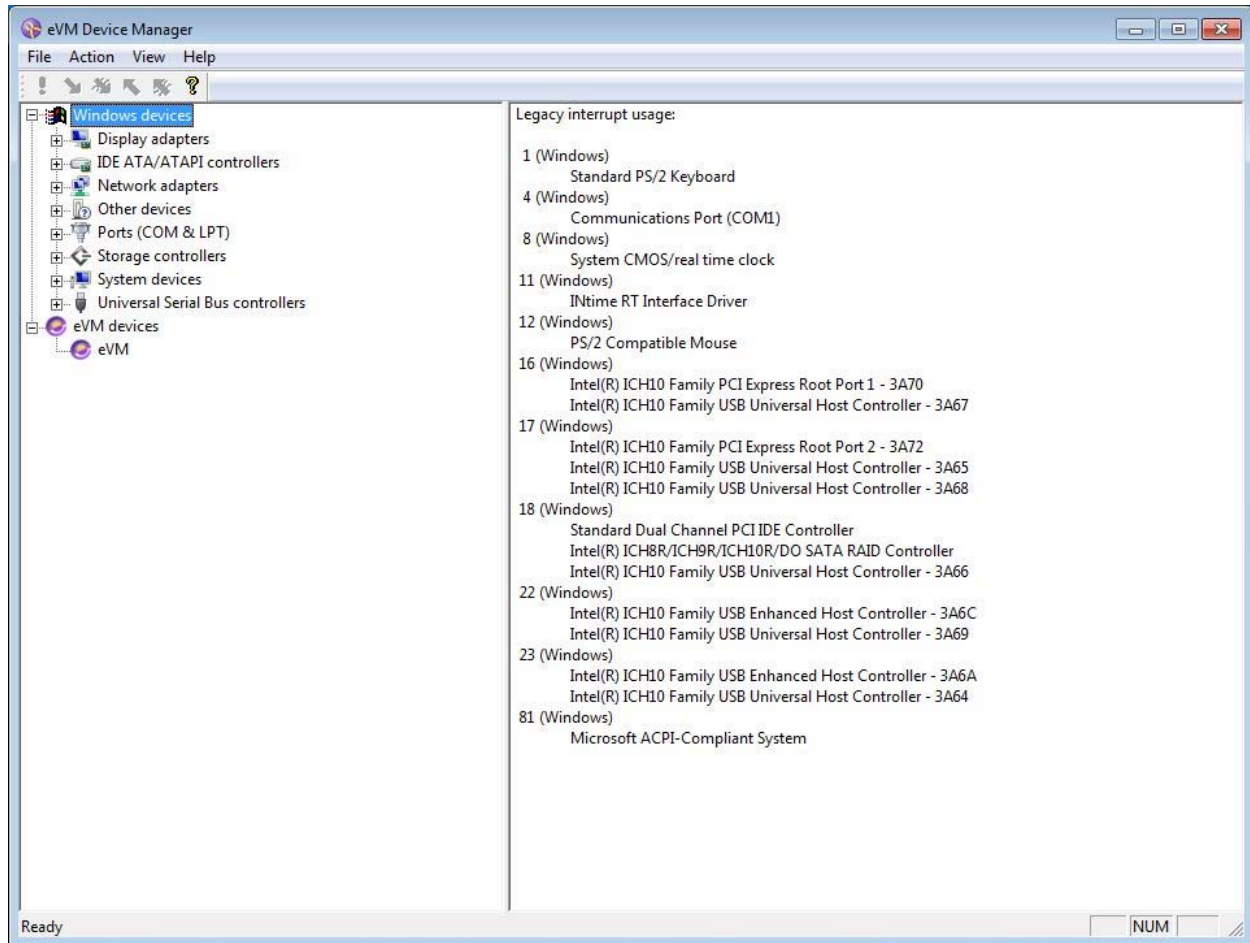


Figure 4.41 - eVM Device Manager

You can use the View menu to select between showing only devices that can be passed or all devices. Figure 4.42 shows eVM Device Manager with only the devices that can be passed being displayed; and when you select the **View** dropdown, you have *All devices* as one of the choices.

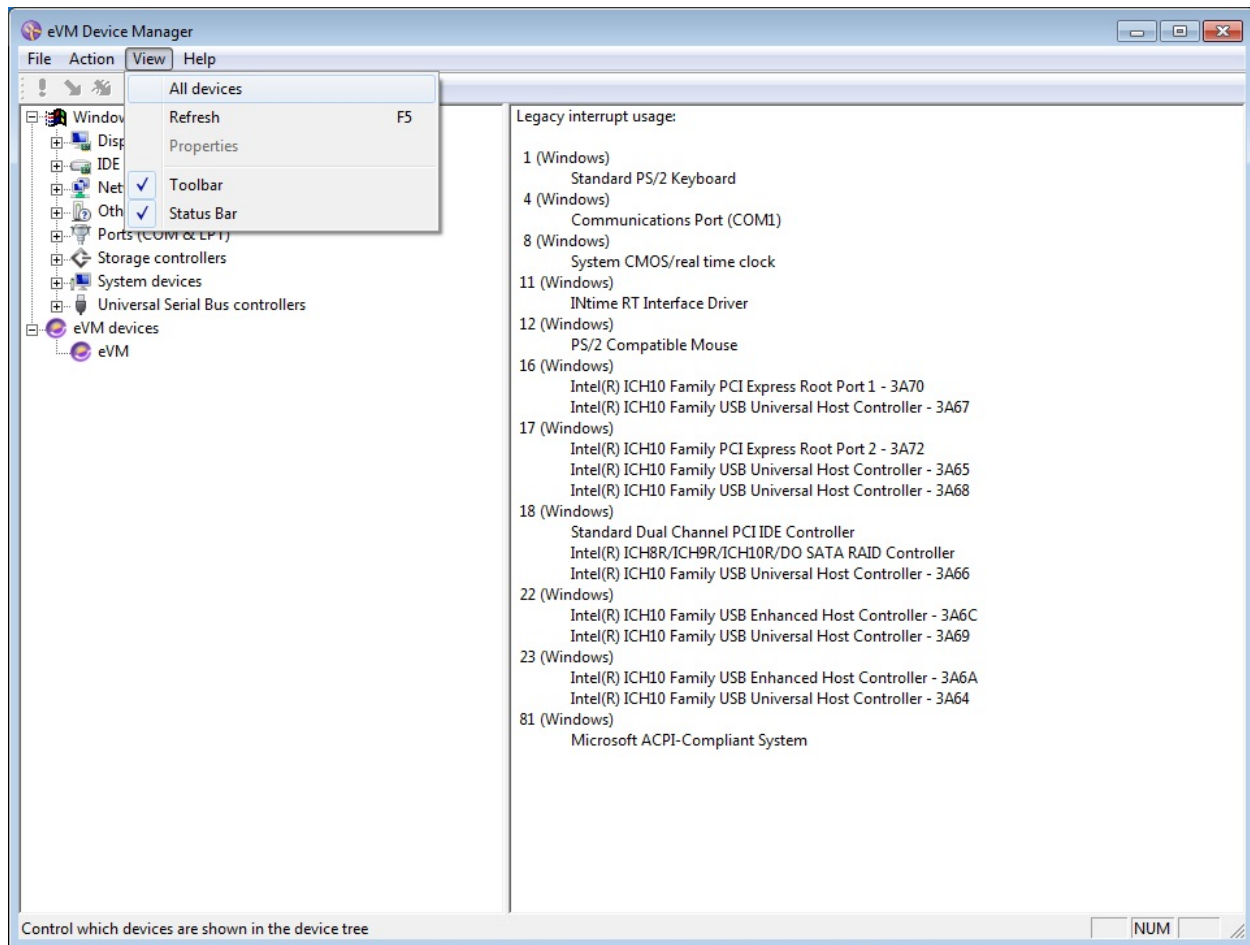


Figure 4.42 - eVM Device Manager Showing Devices that can be Passed

Figure 4.43 shows eVM Device Manager with all devices being displayed, and when you select the **View** dropdown, you have *Devices that can be passed* as one of the choices.

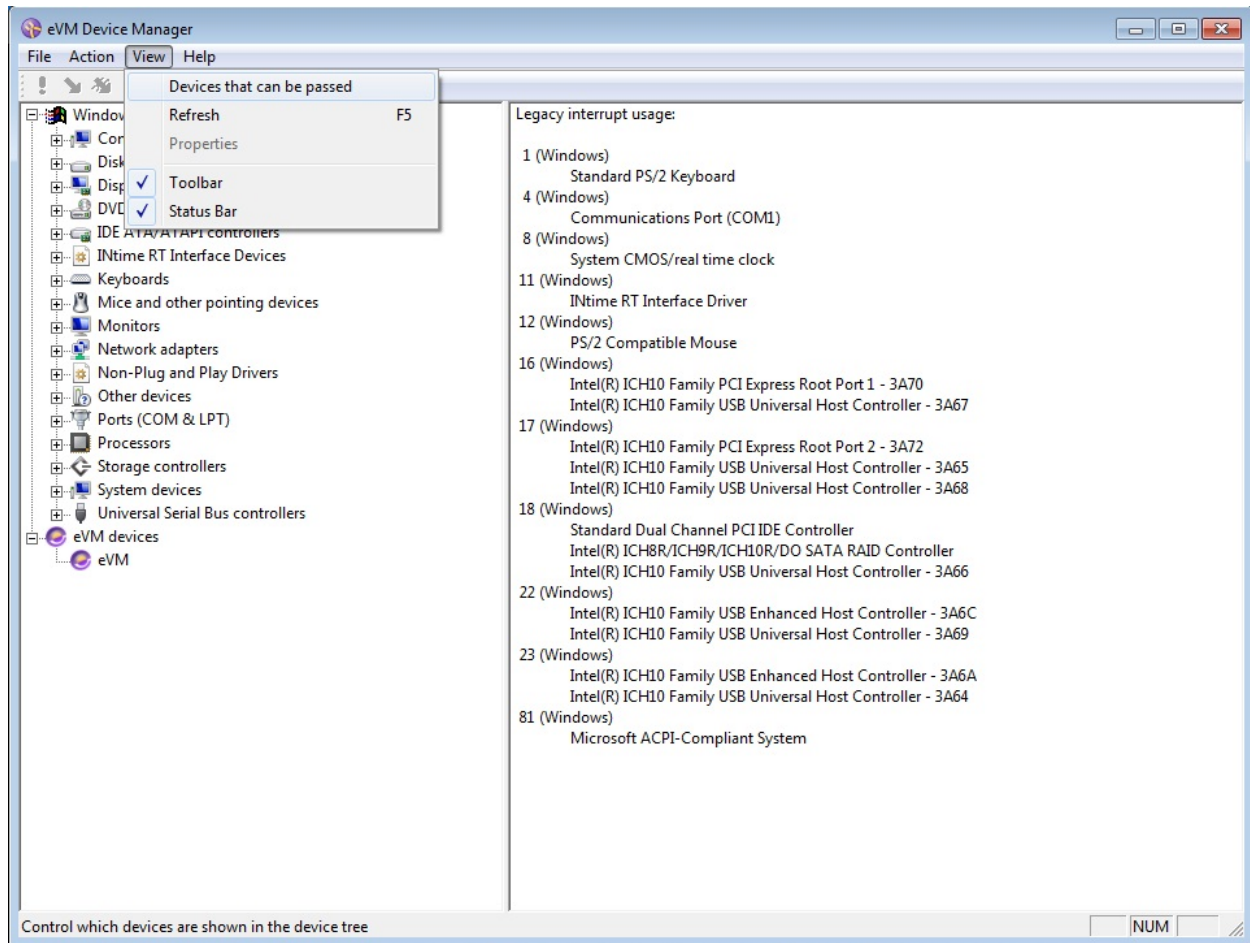


Figure 4.43 - eVM Device Manager Showing All Devices

4.6.2 Passing a Legacy IRQ Device to eVM

For this example, let's choose COM1 as the legacy IRQ device. To confirm that this device fits the criteria, from the dropdown select **View->Devices that can be passed** and look at the right pane and see if COM1 is listed as a device with *Legacy interrupt usage*. Then look at the left pane and expand the Windows devices **Ports (COM & LPT)** device node to see that COM1 is a Windows device that is a device that can be passed to eVM.

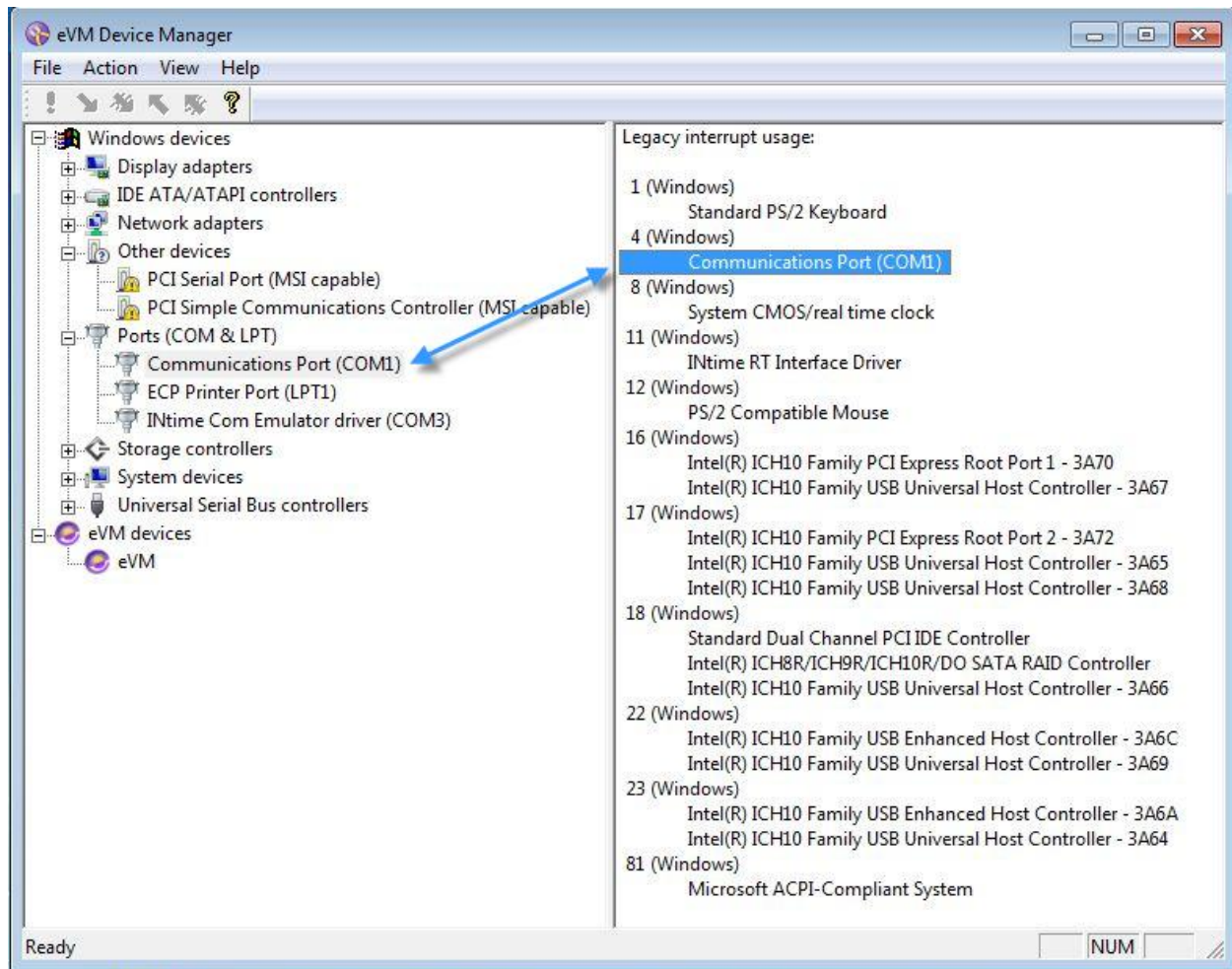


Figure 4.44 - COM1 properties in eVM Device Manager

Figure 4.44 shows that COM1 is a Windows device that can be passed and is a legacy interrupt device.

In the left pane, click on the **Communications Port (COM1)**, and then either click on the **Pass to eVM** button, Figure 4.45.

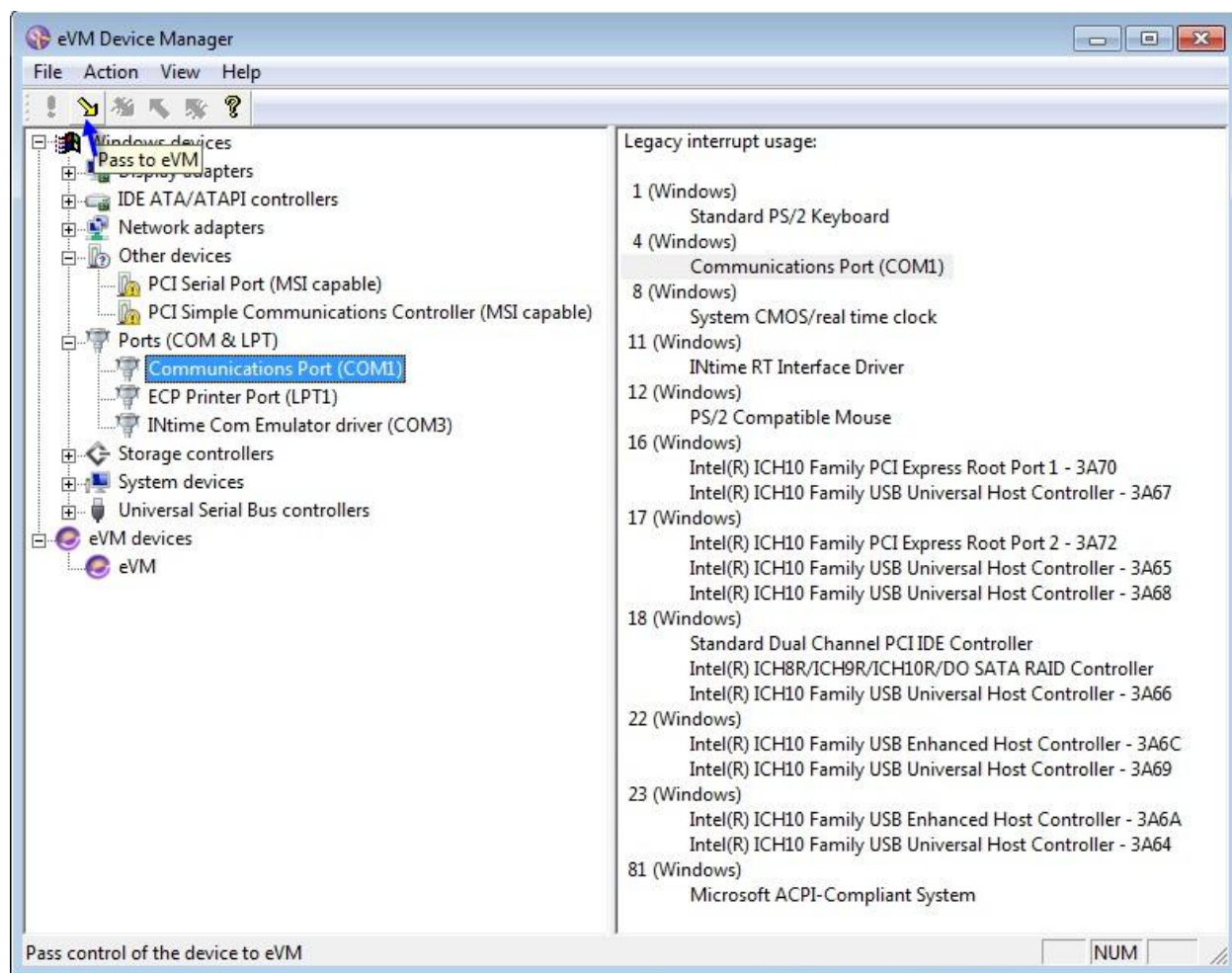


Figure 4.45 - Pass Device Using Pass to eVM Button

Or, from the dropdown, select **Action->Pass to eVM (with legacy IRQ)**, Figure 4.46.

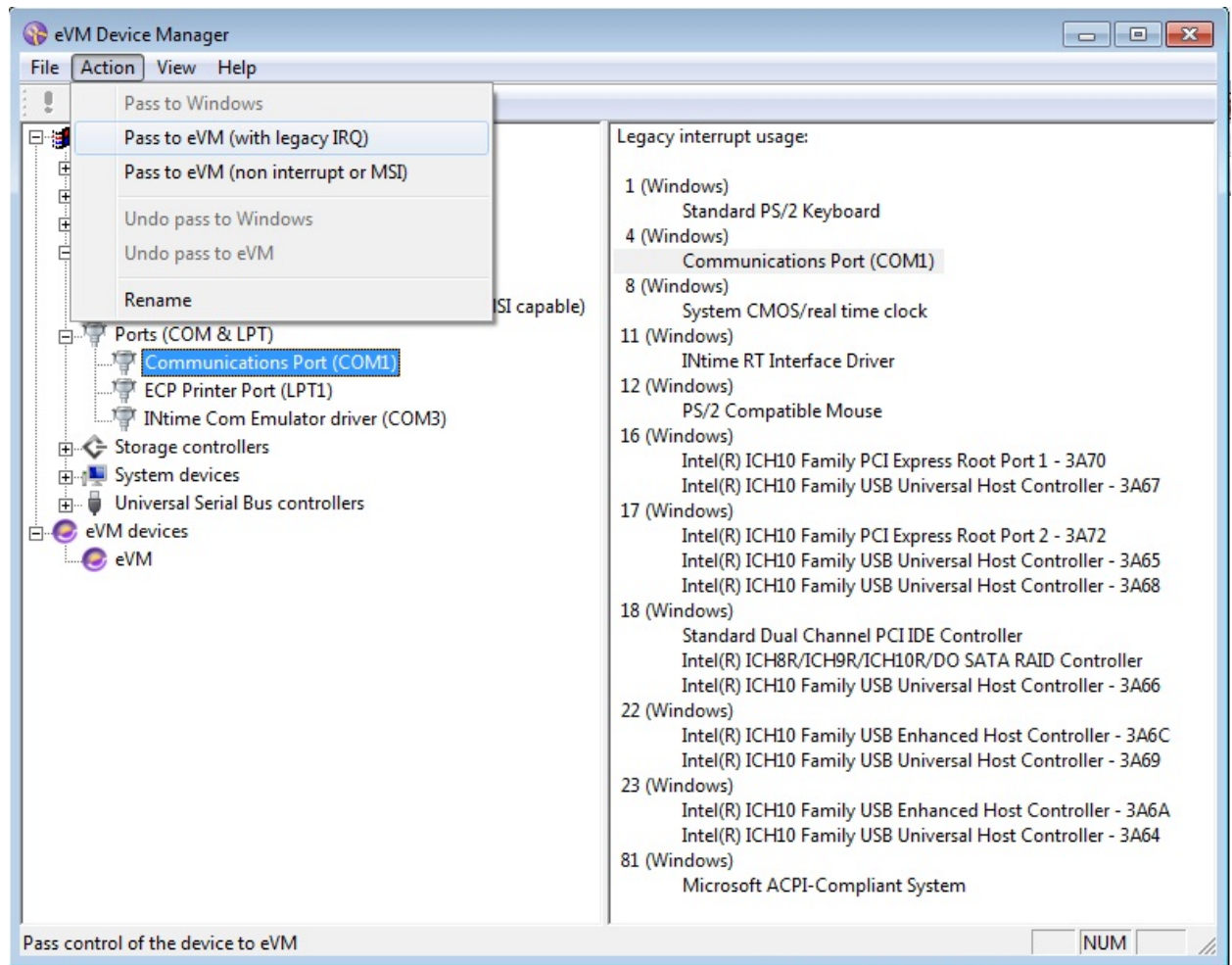


Figure 4.46 - Pass Device Using Action Dropdown

When you select the device to pass, the right pane will show that you have selected COM1 to be passed, but at this point, the device configuration has not changed, Figure 4.47.

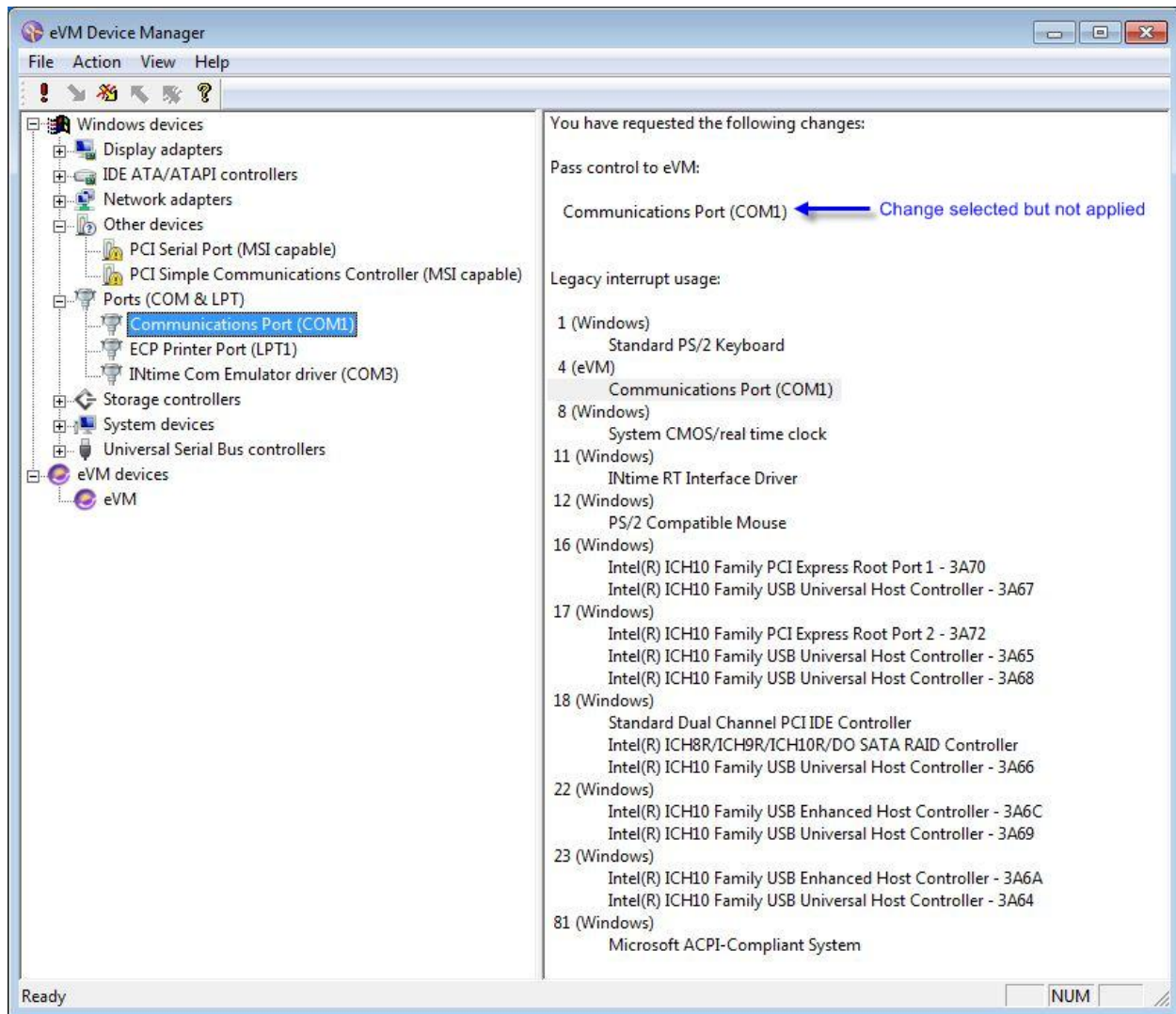


Figure 4.47 - Device Pass Request Selected

To actually apply the requested change in device configuration, you must save the configuration. This is done by either clicking the **Save configuration** button, Figure 4.48.

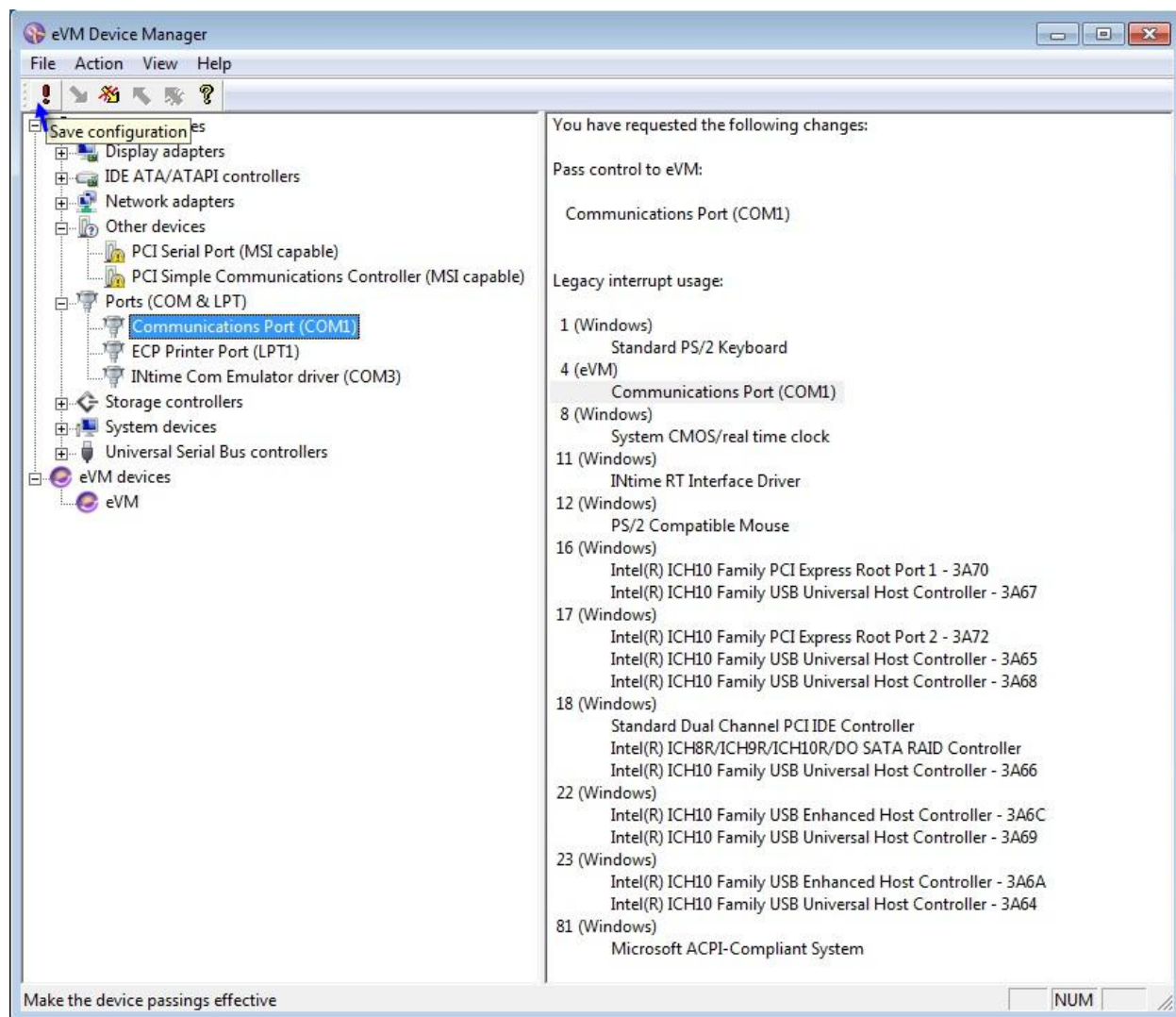


Figure 4.48 - Using the Save Configuration Button

Or, from the dropdown select **File->Save the configuration**, Figure 4.49.

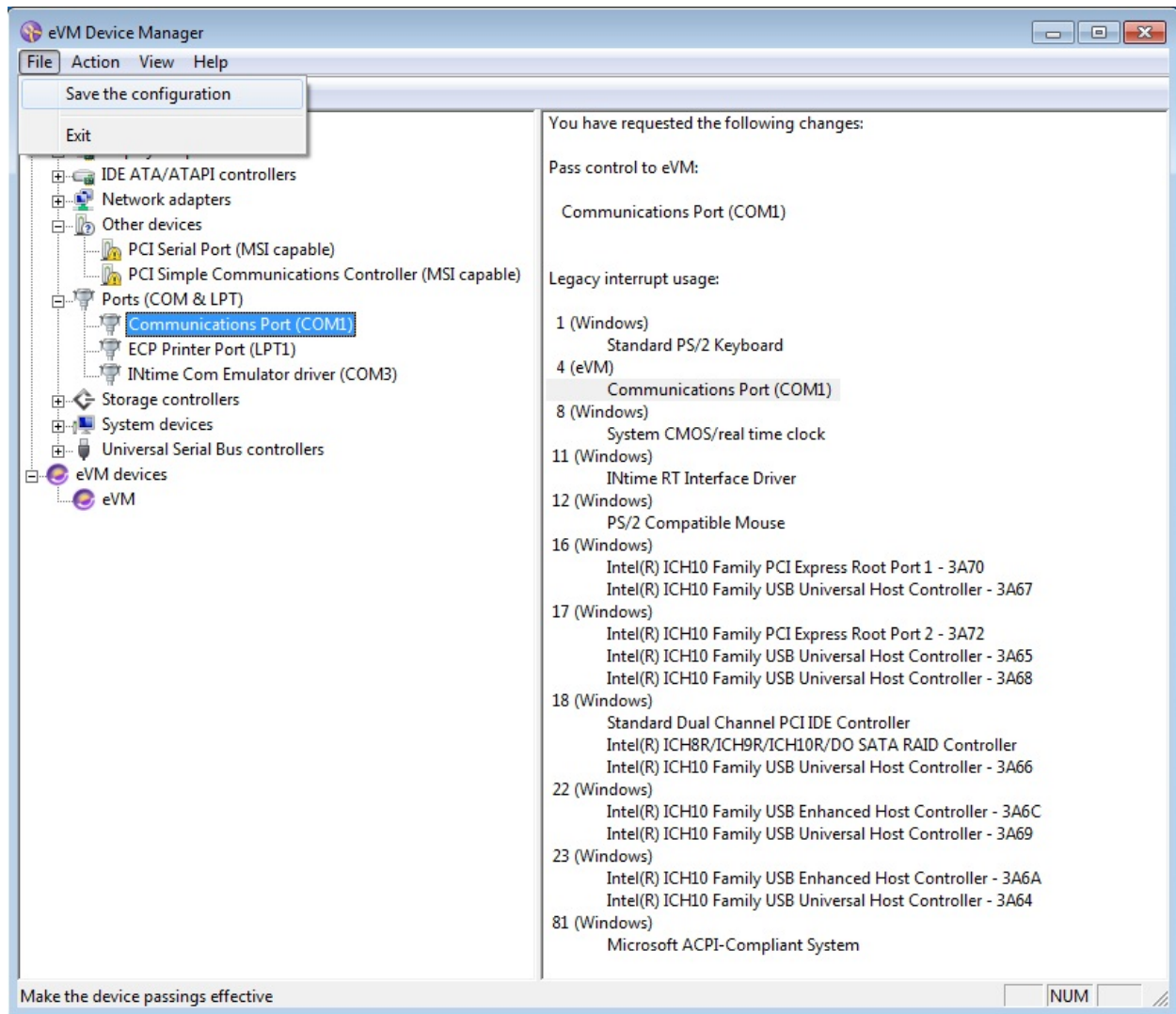


Figure 4.49 - Using Dropdown to Save the Configuration

This changes the device configuration, but it will not actually be applied until the platform has been rebooted. eVM Device Manager prompts you to restart the platform, Figure 4.50.

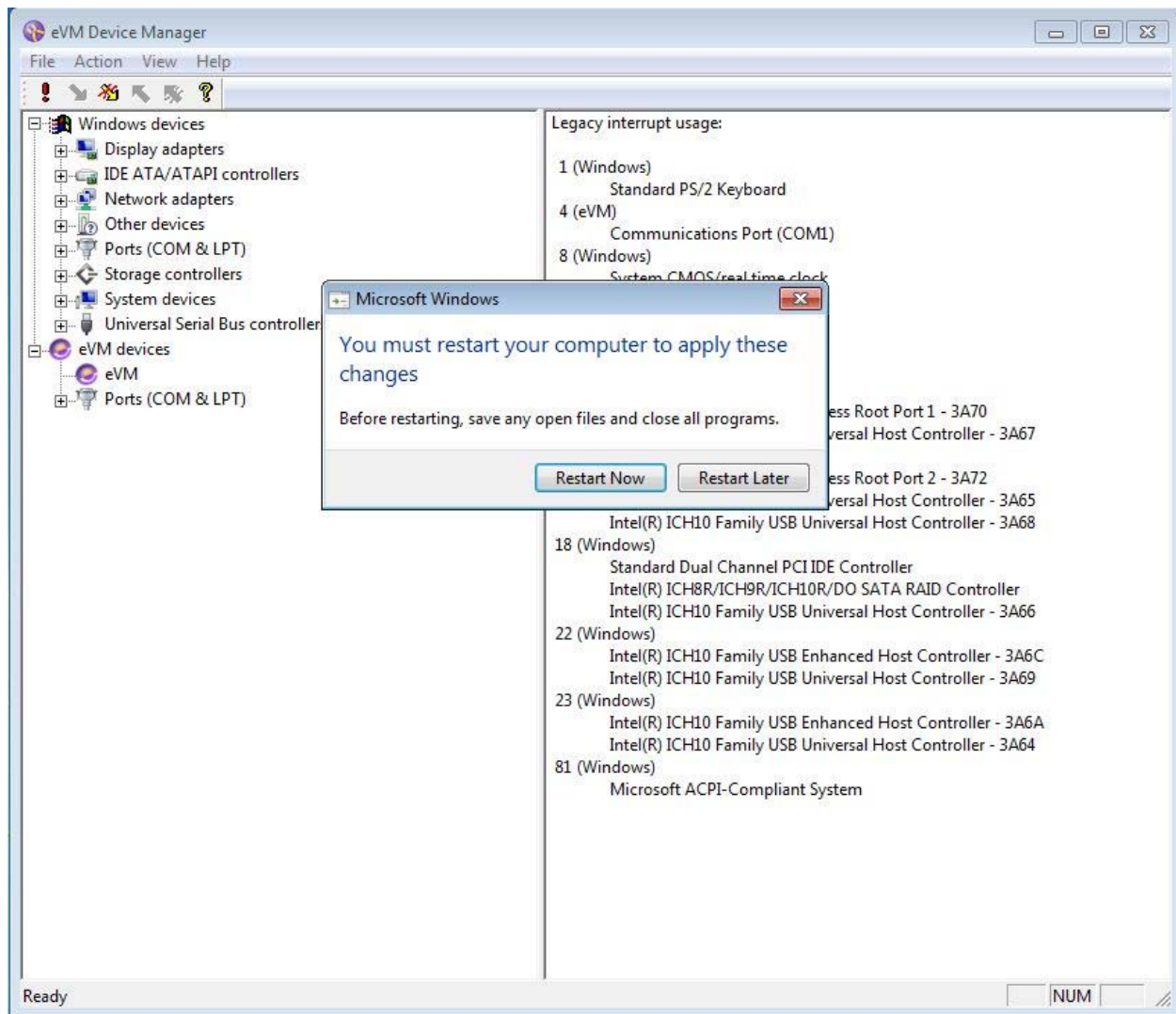


Figure 4.50 - Platform Restart Required to Apply New Device Configuration

When the platform has been restarted, launch eVM Device Manager; and expand the **Ports (COM & LPT)** node under *Windows devices* and under *eVM devices*, Figure 4.51. You will see that the Communications Port (COM1) is no longer listed as a Windows device, but it is now listed as an eVM device.

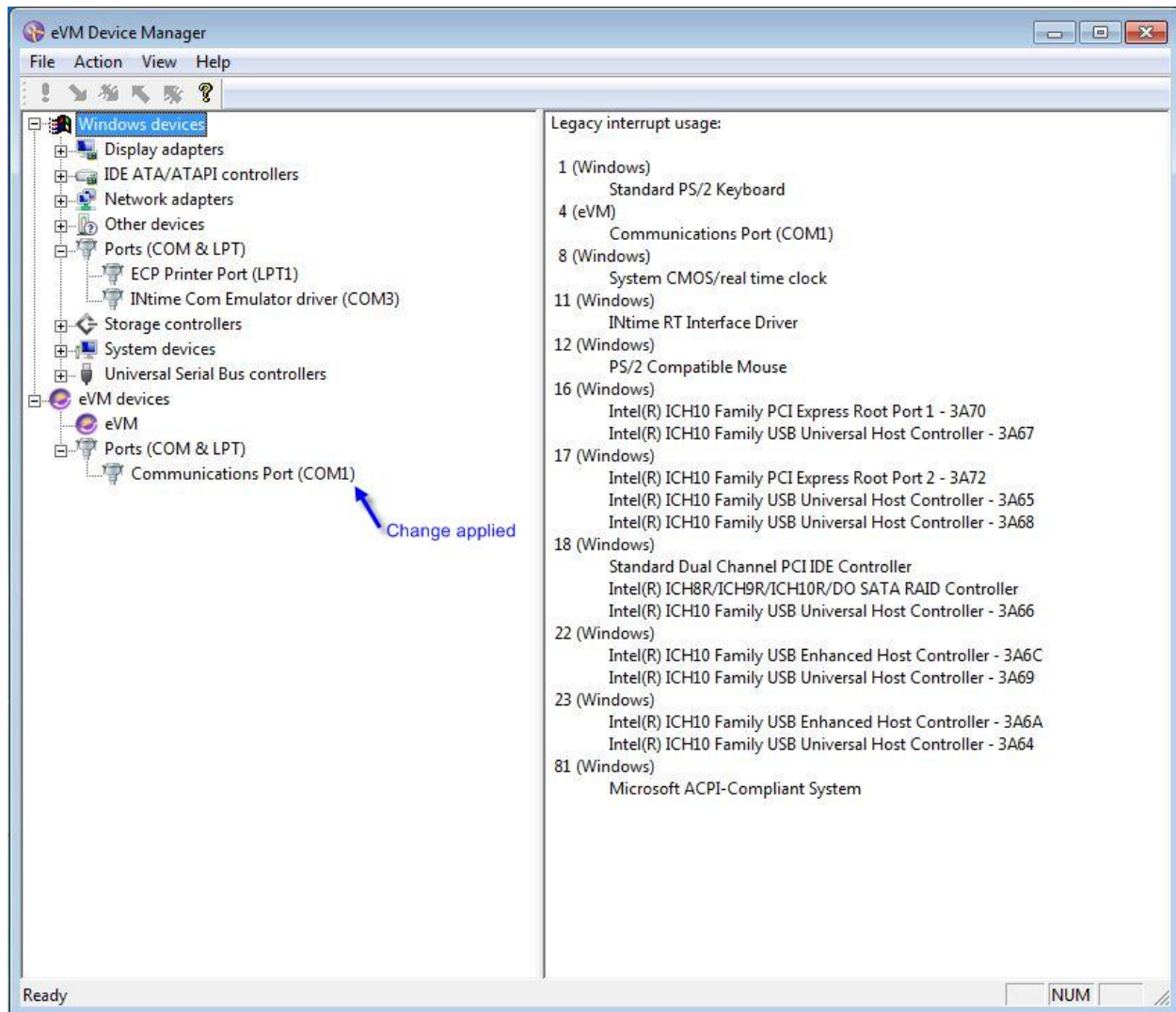


Figure 4.51 - Device Configuration Change Applied

The legacy IRQ COM1 device has now been passed to eVM for exclusive use by the guest OS.

4.6.3 Conflict Example when Passing Legacy IRQ Device to eVM

Remember the issue with sharing an interrupt between Windows and the guest OS? This would make the guest OS dependent on a Windows interrupt handler running to maintain a deterministic system. It is not allowed to assign a device to the guest and a device to Windows that share a common interrupt. The eVM Device Manager will flag an attempt to do this as a conflict.

To illustrate this, open the eVM Device Manager; and under *Windows devices*, expand **Universal Serial Bus controllers**, Figure 4.52.

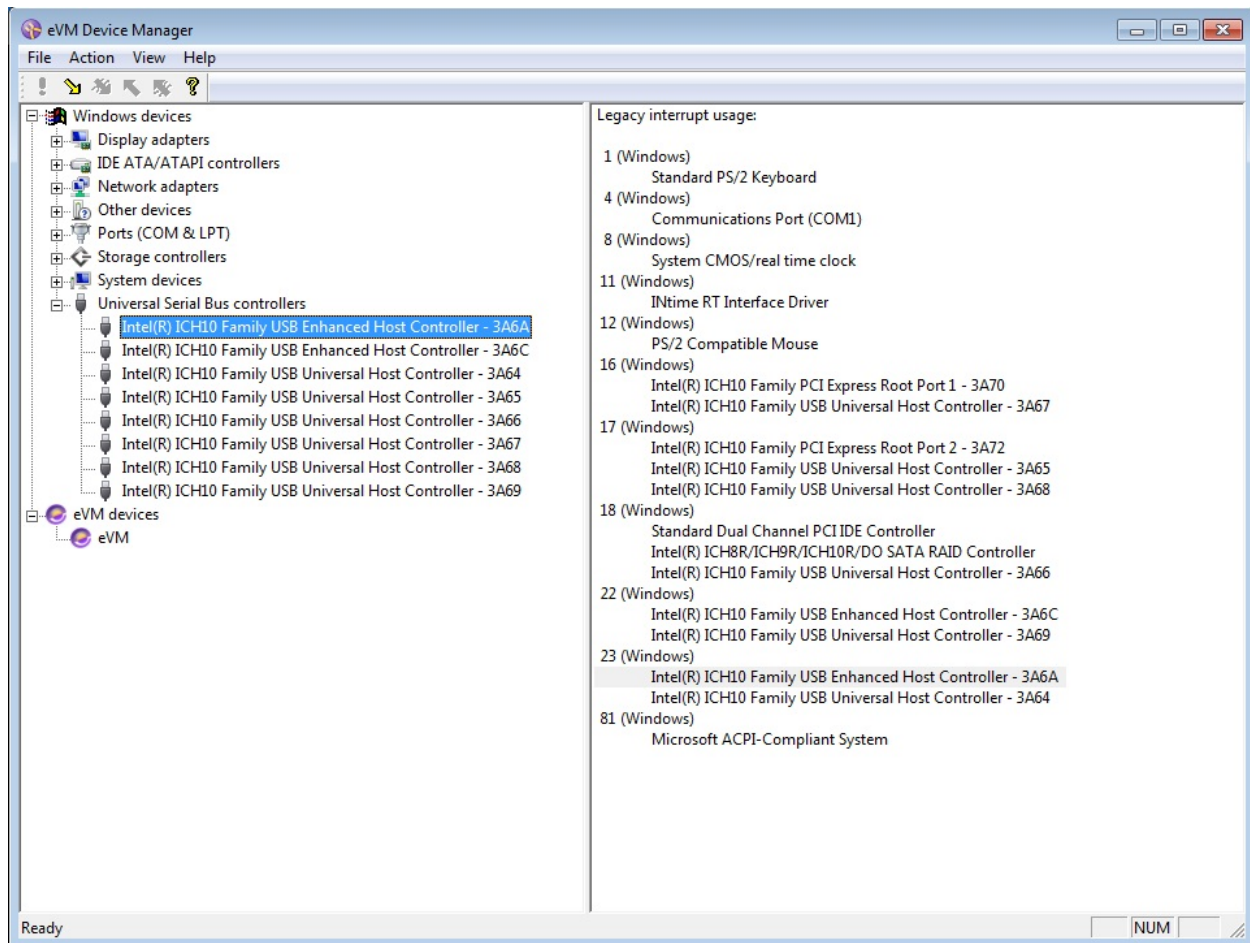


Figure 4.52 - eVM Device Manager Showing Windows USB Controllers

In the left pane, select the first **USB Enhanced Host Controller** and then, from the dropdown, select **Pass to eVM**.

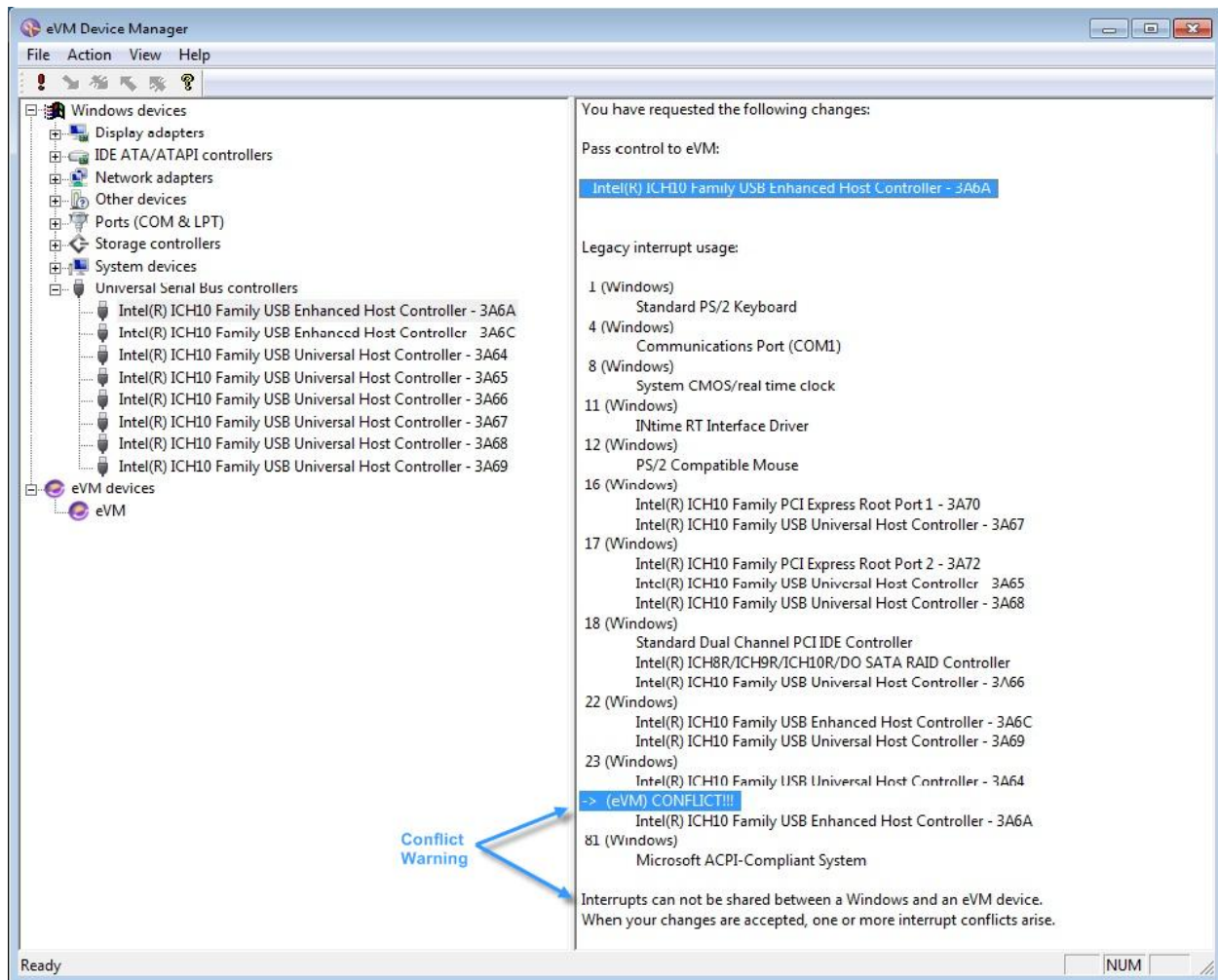


Figure 4.53 - eVM Device Manager Flagging a Conflict

In the right pane, see the conflict warning that is posted by eVM Device Manager, Figure 4.53.

4.6.4 Passing an MSI Device to eVM

In our example system, we have two Ethernet devices. One is an onboard controller that Windows is using, and the other is a PCI Express adapter with MSI capability that can be dedicated to eVM for the guest OS to use. If we expand the Windows devices **Network adapters** node in the left pane of this system, we will see the *Intel® Gigabit CT Desktop Adapter (MSI capable)* device. To illustrate another way to pass a device to eVM, right-click on the **Intel® Gigabit CT Desktop Adapter (MSI capable)** device, and select **Pass to eVM (non interrupt or MSI)**, Figure 4.54.

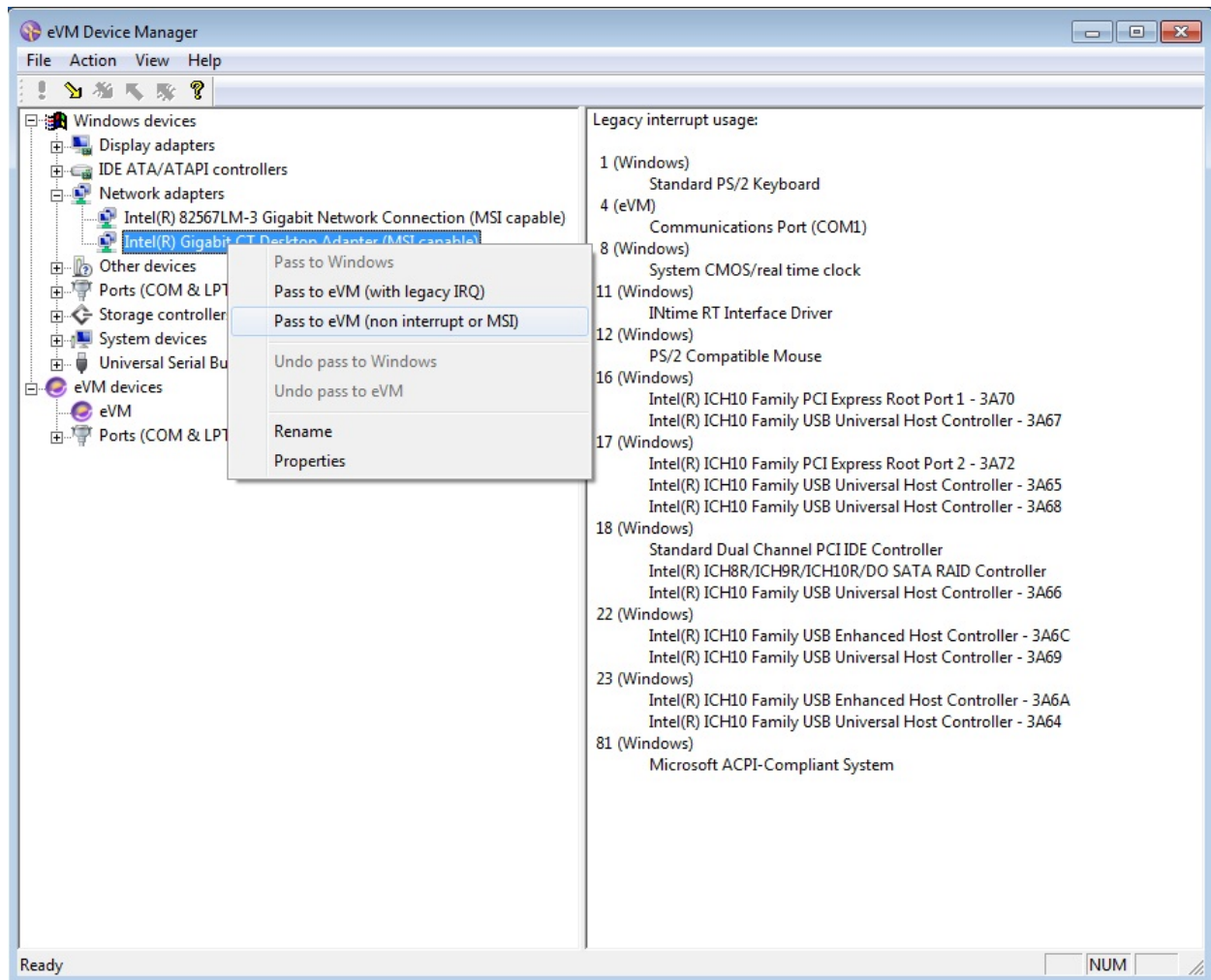


Figure 4.54 - Right-Click a Device to Pass it to eVM

Again, we will see the request to pass the device now listed in the right pane, Figure 4.55.

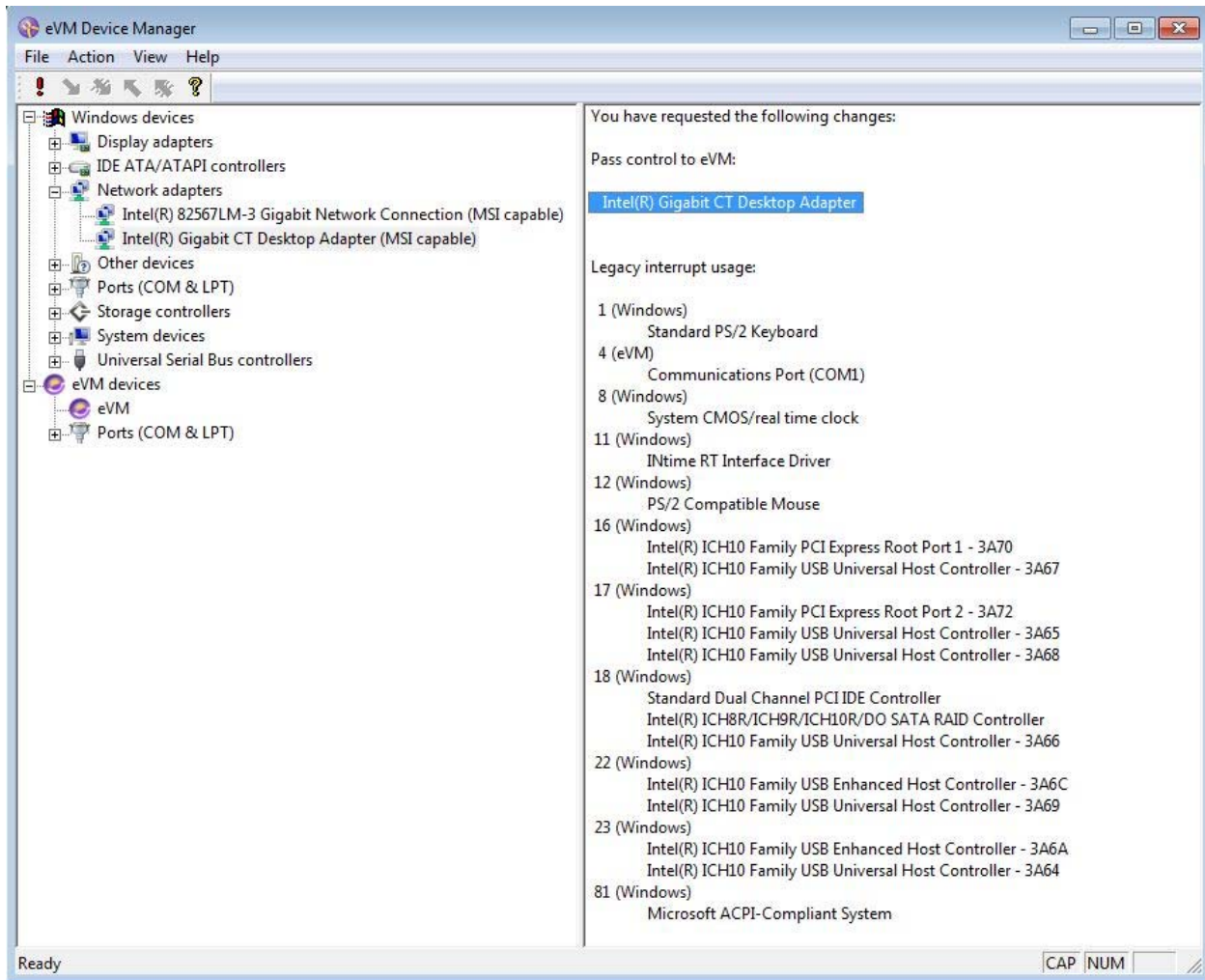


Figure 4.55 - MSI Capable Device Pass Request

Save the device configuration, as before, and restart the platform; and the change will be applied to the platform. After saving the configuration and restarting, launch the eVM Device Manager and expand the **Network adapters** device nodes under *Windows devices* and *eVM devices*; and you will see that the network device has been removed from Windows control and is now dedicated to eVM for use by the guest OS, Figure 4.56.

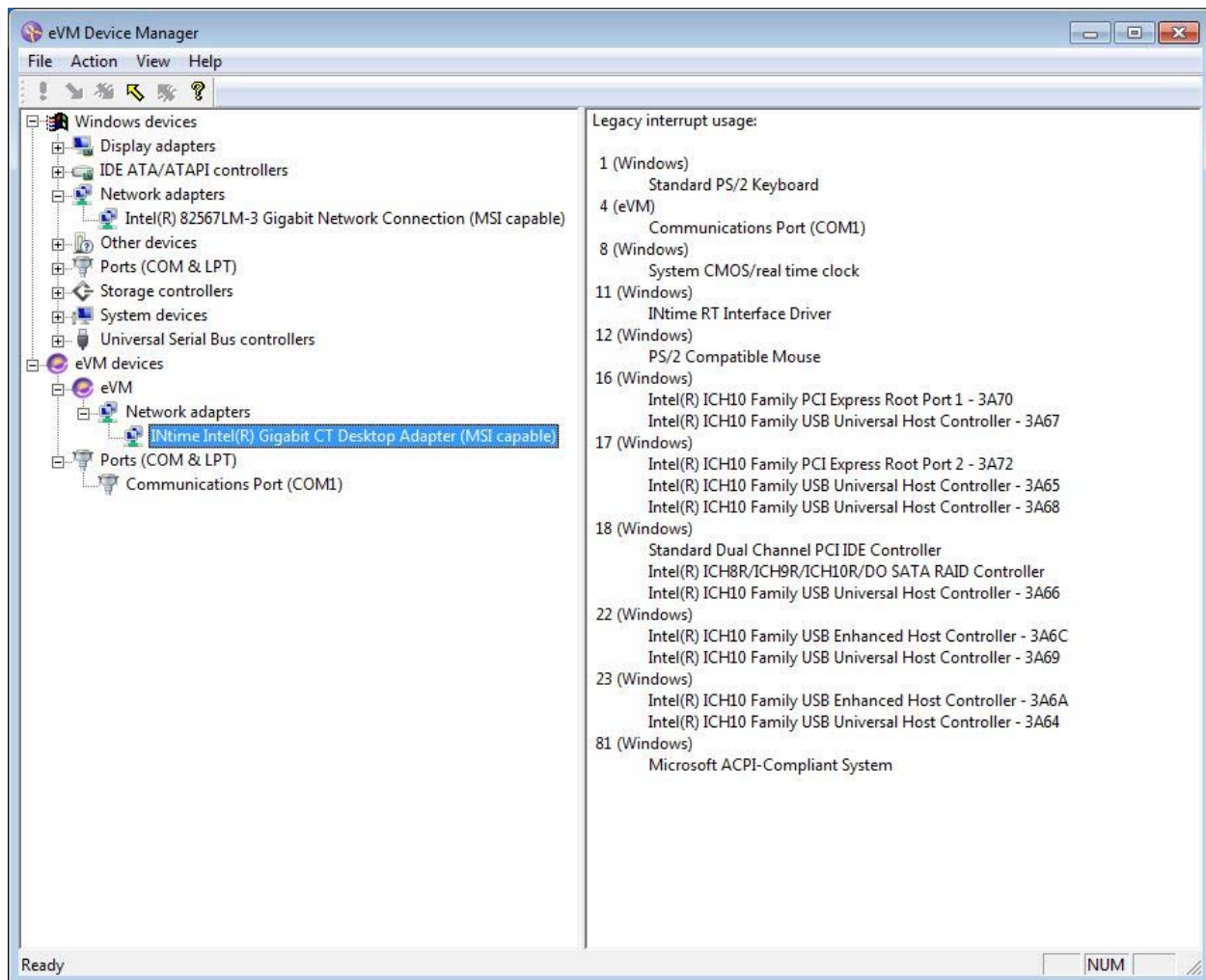


Figure 4.56 - MSI Network Adapter Device Passed to eVM

4.6.5 Bus Master Warning Example

To illustrate the bus master warning that the eVM Device Manager will issue when passing an MSI device to a platform that does not support VT-d, we turned off VT-d support in the BIOS of our test platform, and check using VtProbe that VT-d support had been disabled, Figure 4.57.



Figure 4.57 - VT-x Support Enabled but VT-d Support Disabled

Then we repeated the exercise in Section 4.6.4, passing the MSI network device to eVM. When we select **Pass to eVM**, the eVM Device Manager issues a warning for the network device, Figure 4.58.

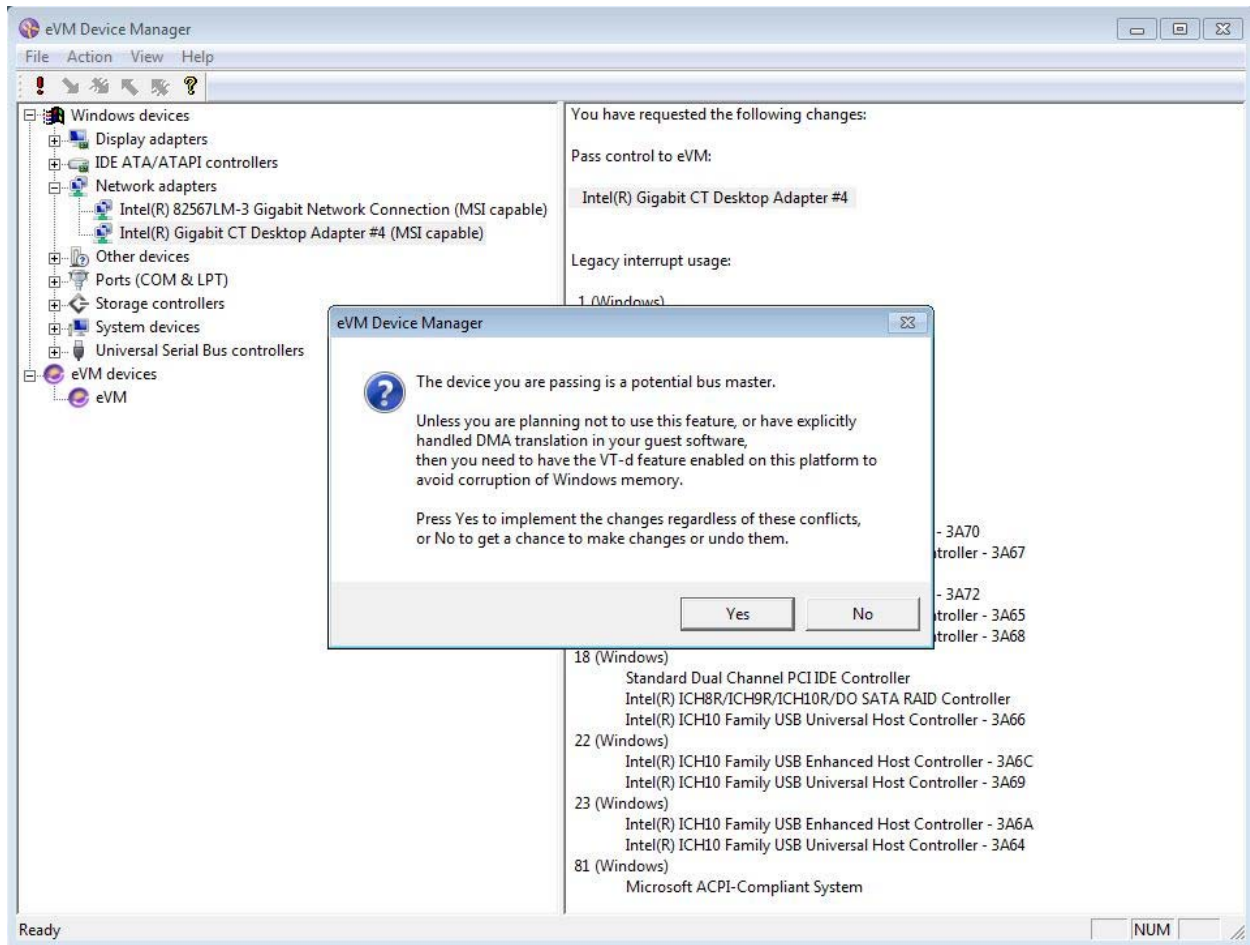


Figure 4.58 - Bus Master Warning for no VT-d Support

If you allow the eVM Device Manager to pass this device to eVM, such a condition would require that the bus master driver be modified to manage the address offset for the virtual machine, or bus master capability would have to be disabled in the device, if possible.

4.6.6 Passing a Device back to Windows

Now, let's give the network adapter that was passed to eVM in Section 4.6.4, back to Windows. Select the **Intel® Gigabit CT Desktop Adapter** in the eVM *Network adapters* node in the left pane of the eVM Device Manager, and click on the **Pass to Windows** button, Figure 4.59.

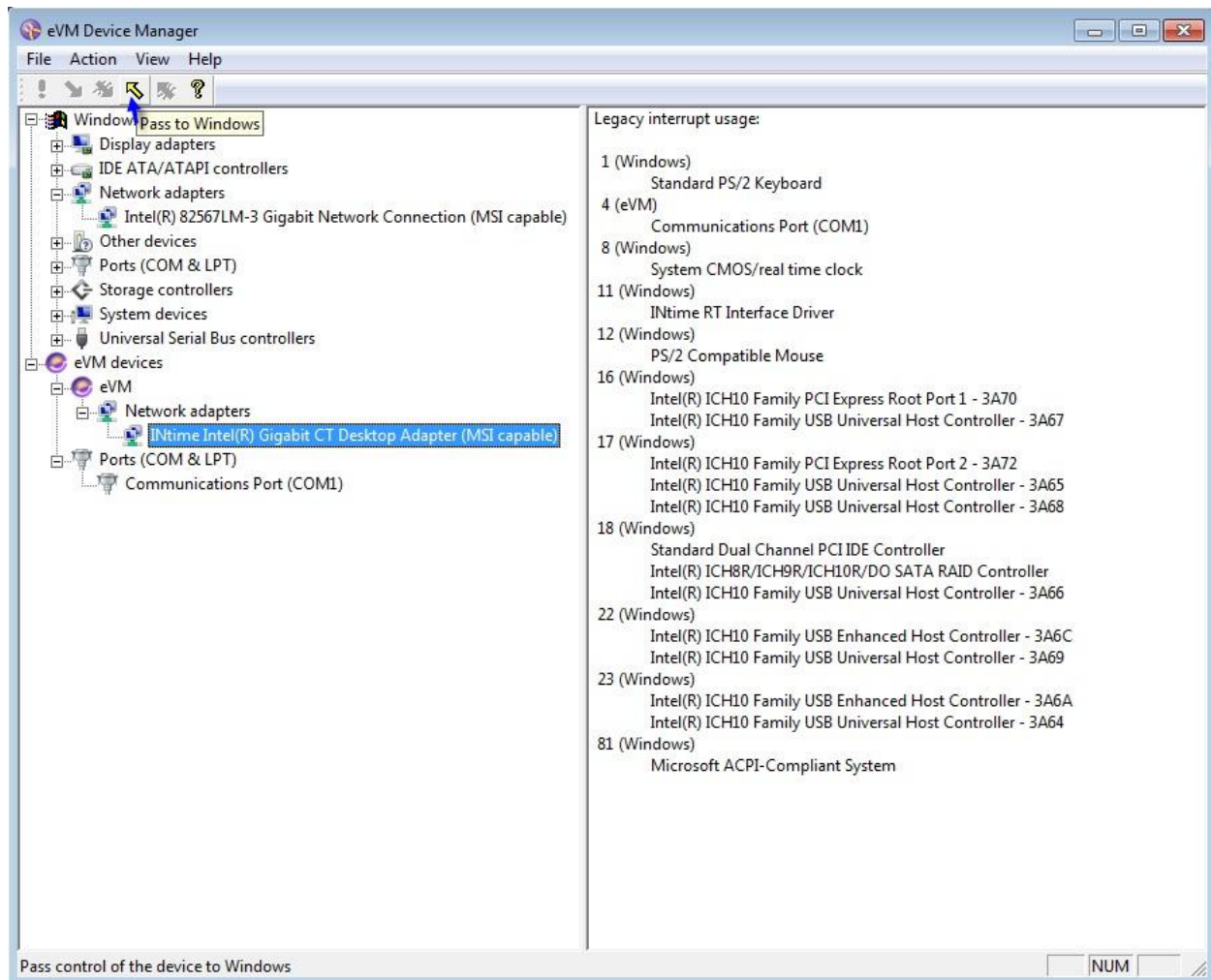


Figure 4.59 - Select the Network Adapter and Pass It Back to Windows

Again, eVM Device Manager will show the request in the right pane, except this time it shows that we are requesting to pass the device to Windows, Figure 4.60.

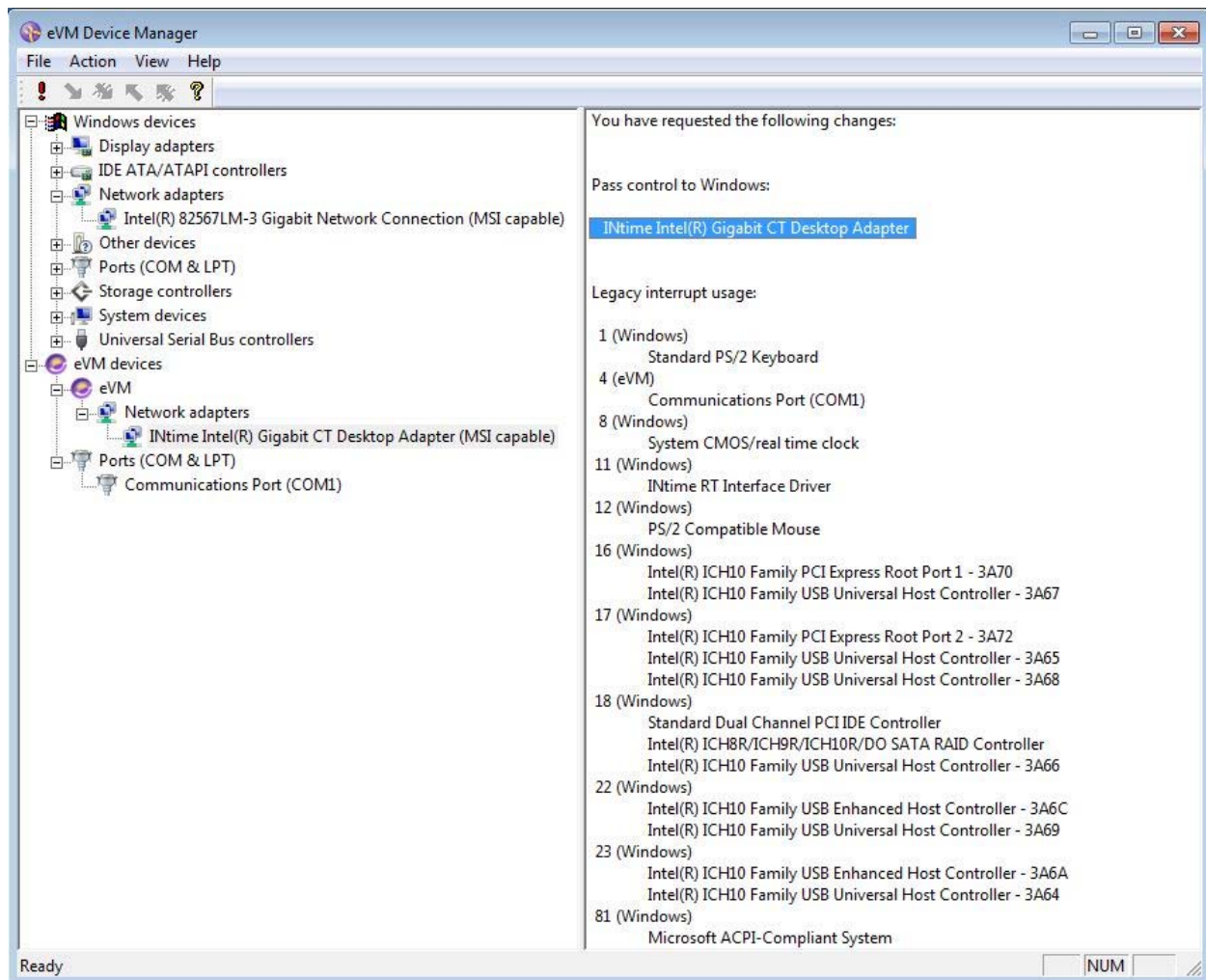


Figure 4.60 - Request to Pass the Network Controller back to Windows

The eVM Device Manager will prompt you to restart the platform to apply this change, Figure 4.61.

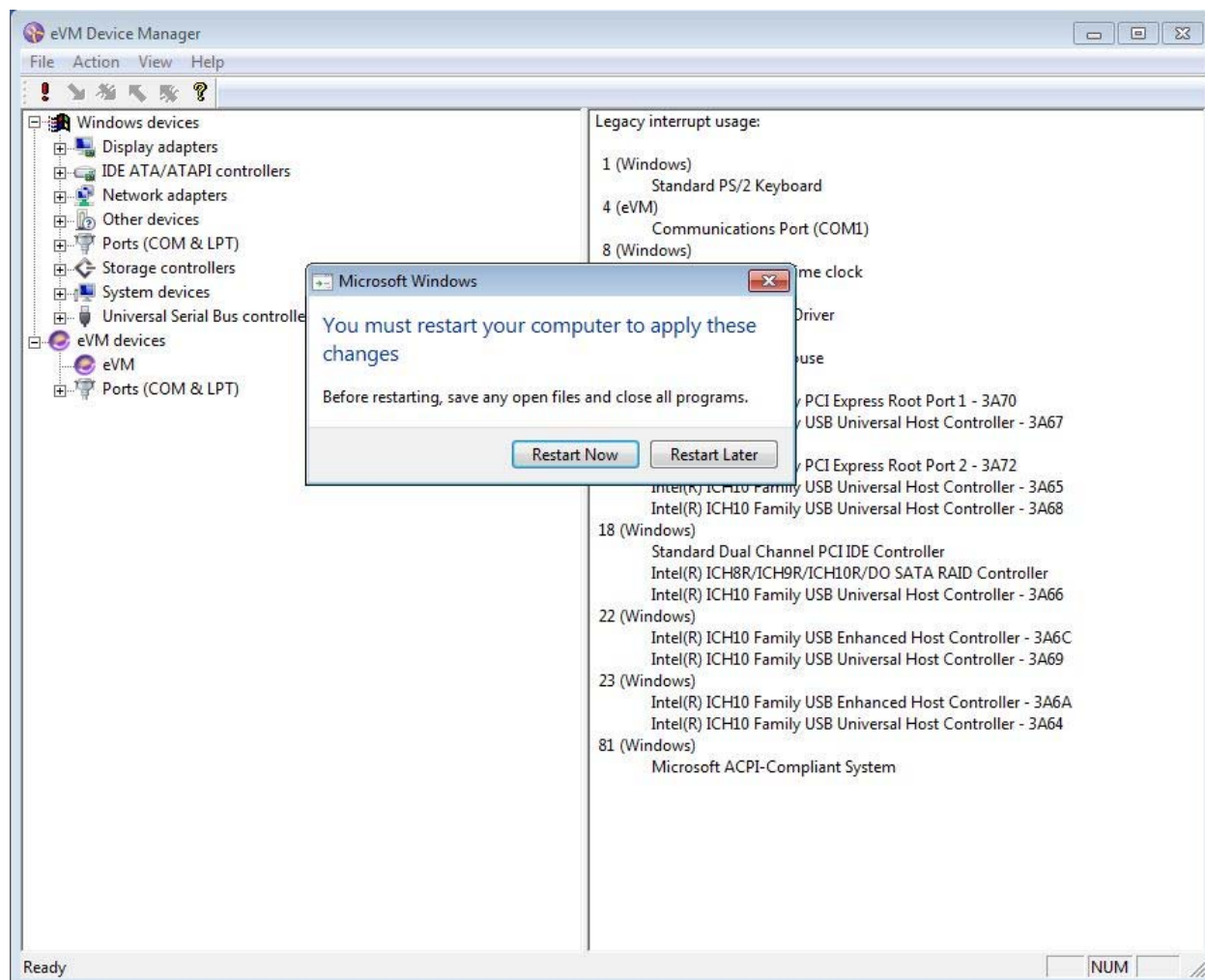


Figure 4.61 - Restart Request to Apply Device Configuration Changes

After restarting the platform, launching the eVM Device Manager, and expanding the device nodes in the left pane, it can be seen that the network adapter is back under Windows control, Figure 4.62.

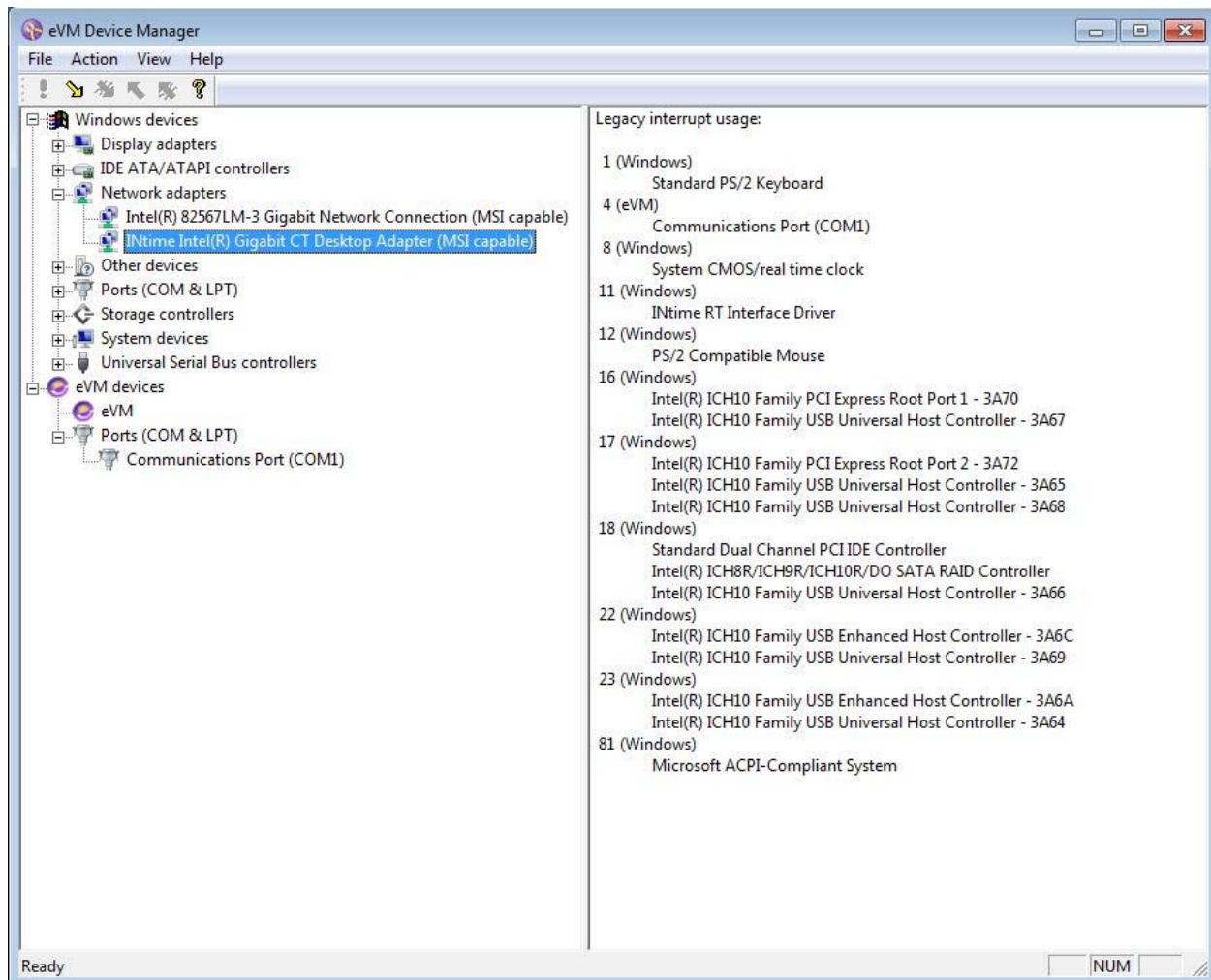


Figure 4.62 - Network Adapter Back Under Windows Control

A. Production installation

After you have set up eVM to meet your application requirements on your target development system, you are probably going to want to replicate that setup in a production environment. To do that without having to configure each and every installation of eVM in production, you can save the configuration from the target development system that is already set up and import that saved configuration to each of the production systems.

This operation is done with the **evmconf.exe** utility that has been loaded as part of the eVM for Windows installation. This utility can be executed from the Windows **Run** command by pressing and holding the **Windows** key and then hitting the '**R**' key. In the Run dialog you can enter the command line parameters described below, or you can also launch the **CMD** Shell from the Windows **Run** command and enter the command line parameters described below at the command prompt. In Windows Vista or Windows 7 you can also enter the command line parameters described below in Windows **Start Search**.

Note: for the "/n" command line switches, 'n' is case insensitive.

A.1 Saving the configuration with the /e: option

Syntax usage:

evmconf /e:[export file name and full path]

Once you have configured eVM on your target development system, the /e option of **evmconf.exe** will allow you to export the known working configuration to a file that can be used later to import into other systems to configure them to the same setup parameters as the target development system that this configuration was exported from, or it can also be used to restore the eVM configuration of your target development system to this known state, if you have made configuration changes after saving this configuration.

In the example below we have a system that is already configured whose configuration has been labeled "My Old OS"

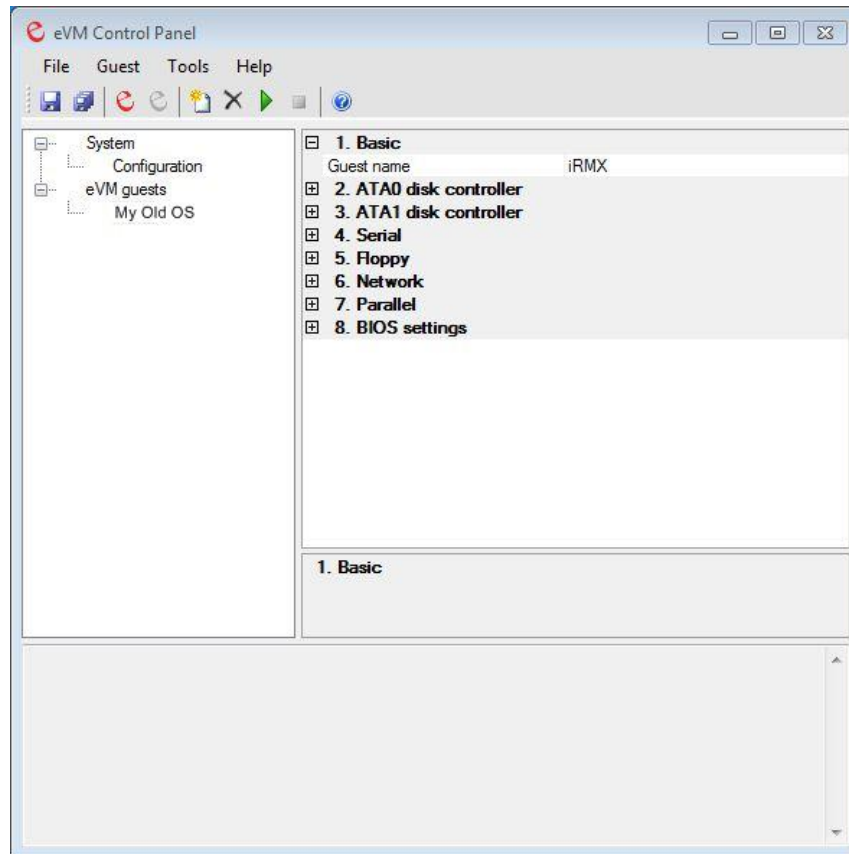


Figure 4.63 - eVM Configuration for My Old OS

We are going to save that configuration in a file named `MyeVMConf.ecf` by, first closing the eVM Control Panel and then entering the command shown below from the **run, CMD Shell**, or **Start Search** command line.

evmconf /e:c:\MyeVMConf.ecf

This will create a file called ***MYeVMConf.ecf*** in the root directory of the C: drive.

Note: any file extension can be used. The ".ecf" extension is the default.

A.2 Restoring or importing a known configuration

When you first install eVM, there are no settings included in the configuration, as shown in the screen capture Figure 4.64 below. The ***/i*** option of the ***evmconf.exe*** command allows you to import a known setting from another system and allows you to restore a saved setting to a modified configuration.

Syntax usage:

evmconf /i:[saved configuration file name and full path]

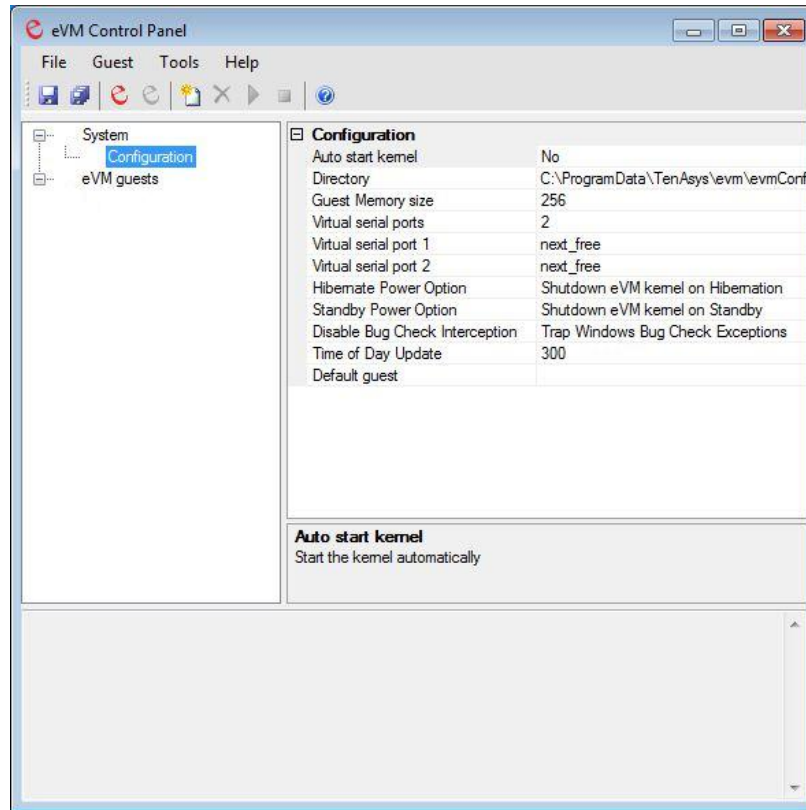


Figure 4.64 - eVM Configuration

To import the settings from the target development system to a new production system, first close the eVM Control Panel.

Second, copy the configuration **MyeVMConf.ecf** file to the root directory of the C: drive of the of each new production system.

Then import the configuration by entering the command shown below from the **run, CMD** Shell, or **Start Search** command line.

evmconf /i:c:\MyeVMConf.ecf

Opening the EVM Control Panel on the production system should display a configuration labeled **My old OS** with the settings of the imported configuration. If this was used to restore the configuration of a modified system, the EVM Control Panel should show that the modified system settings have been restored to the **My Old OS** settings.

A.3 Assigning devices to eVM on the command line

PCI devices

When the guest OS needs full control of a PCI device, the PCI device should be passed to the guest. This can be done from within the eVM Device Configuration utility, or it can be done from the command line:

DevConfig /option "device" [/instance n]

Where 'option' is either of:

Toevm - pass a device to eVM and let the guest use the interrupt as defined in the current configuration. The interrupt should not be shared with another device that is controlled by Windows.

Toevmnoirq - pass a device to eVM and let the device use MSI (Message Signaled Interrupts); a unique interrupt number will be assigned.

"device" - must be the full device name as shown in the eVM Device Configuration utility under the eVM tree. You may replace trailing characters by a single star (*), if that still leaves a unique device name.

When more than one device matches the full device name, an instance number must be specified: zero indicates the first instance.

A.4 Command Line Installation of Virtual Serial Ports

When the guest OS needs one or more virtual serial ports, these can also be installed from within the eVM Control Panel or from the command line:

evmconf /s:[number]

Where 'number' is the number of virtual serial ports you want to install (from 0 to 4).

The configuration result is the same as opening the eVM Control Panel; and in the System Configuration properties, setting Virtual Serial Ports to 'number', setting each virtual serial port to 'next_free', and then saving the system configuration. Using '0' will remove all virtual serial ports.

A.5 Debug logging

When evmconf is running, debug information is presented in the bottom pane of the window. You can choose to save the session's logging to a file, evmconf.log, in the %EVMCONF%

directory tree. Please note that the file will only save the current session, and a new session will start with a new blank file.

evmconf /l

You can also log additional information to the file with the /v, verbose, switch

evmconf /l /v

B. eVM Virtual Machine

The TenAsys eVM product presents a virtual machine to guest software within a Windows environment which emulates an Intel x86 based PC with certain chipset and other characteristics. In order to run eVM there is at least an Intel dual-core platform required that supports VT-x. The specification for the virtual machine created by eVM is described as follows.

B.1 System Requirements

The minimum requirement to run eVM is a PC platform running Windows XP, Windows Vista or Windows 7 with an Intel dual-core processor or single-core processor with hyperthreading and VT-x supported by the CPU and enabled in the BIOS.

B.2 Base Virtual Machine Specification

The base platform consists of the following components:

CPU

One core of the platform processor is used for the virtual machine. The other core or cores are used by Windows. Most of the native characteristics are available but some features are hidden, such as support for VT-x (VT-x features are not supported in the guest operating system). CPUID instructions are intercepted and some return values modified to hide features. The registers modified are the four returns by CPUID with EAX=1 and all bits are configurable internally. By default the following bits are disabled (always return 0) in ECX: VMX_CAPABLE (bit 5). The following are disabled by default in EDX: HTT, MTRR, APIC, PAE

Memory

Memory is allocated as a single physically-contiguous region to the guest software, and the VMM takes care of translating the physical addresses generated by the guest software.

Paging is emulated using the VTLB algorithm defined by Intel in this version of the VMM. Systems which do not perform well under the VTLB algorithm are demand-paging systems which frequently reload the page directory pointer (such as Windows or Linux). As VT-x2 hardware features become available this implementation will be superseded, resulting in much improved performance for such guest operating systems.

Hardware generated memory addresses (from a PCI bus master device for example) will be automatically translated if the base platform supports and enables VT-d technology; otherwise a change will need to be made to any software, such as device drivers, which calculate a physical address. An interface is provided to the guest software by the VMM which generates the requested offset when making such a calculation.

Interrupt controller (8259A PIC)

The standard interrupt controller is the twin 8259-PIC architecture found in the legacy PC platform. This is emulated in the VM. All modes are supported except non-8086 mode and Special Fully Nested Mode.

The default interrupt map for the VM is as follows:

IRQ	Usage
0	System timer interrupt (from 8254)
1	(reserved)
3	Serial port 1 (COM2)
4	Serial port 0 (COM1)
5	PCI IRQ or
6	Floppy disk controller, or PCI IRQ
7	Spurious interrupt, or PCI IRQ
8	CMOS/RTC
9	PCI IRQ
10	PCI IRQ
11	PCI IRQ
12	NE2000
13	Legacy FPU error
14	Primary IDE controller
15	Secondary IDE controller

Table B 1 Virtual IRQ's

APIC mode will also be supported in an early update of the VMM.

System Timer (8254 PIT)

The system timer is the standard 8254 Programmable Interval Timer found in the legacy PC platform. It is emulated in the VM. Modes 1, 4 and 5 are currently not supported, nor is any BCD counting mode or 8-bit counting modes. Port 61 bits 1 and 4 are functional (speaker data and refresh timer bits).

CMOS & Real Time Clock

The CMOS access ports 70 and 71 are fully supported, and a CMOS RAM image is built at guest boot time from the eVM configuration data. The real-time clock is accessible through the CMOS ports in the normal way, and it will generate IRQ 8 as expected.

Port 92

Port 92 bits 0 ("high-speed reset") and 1 ("A20 control") are supported. Other port 92 bits are not currently supported.

Serial ports (16550 UART)

An emulation of the standard 16550 UART is supported by the VMM and up to four serial ports are configurable. All modes of the 16550 UART are supported. Accesses to the virtual device may be either passed through to a real UART or else the virtual UART transmits and receives data to and from a Windows virtual serial driver, so a virtual serial link may be established between Windows and the guest software.

DMA controller (8237 DMA controller)

The legacy 8237 DMA controller pair is virtualized by the VMM. Currently the only device supported by the virtual controller is the virtual floppy disk controller.

Floppy Disk Controller

This device simulates a standard floppy disk controller. Two devices are supported (drives A: and B:) and only 1.44 Mbyte diskettes are currently virtualized. The media itself is a disk image of floppy disk medium, served by a Windows process to the VMM.

IDE controller

The standard IDE controller is virtualized by the VMM and presents two IDE controllers (primary and secondary) to the VM. Each virtual disk is actually a device image presented by the VMM from a Windows process. Virtual hard disk drives and CDROM devices are supported. The device image takes the form of a binary file. There will also be support for access to whole devices such as disks, or FLASH memory devices, in an early update of the VMM.

PCI Host Bridge (440FX Host Bridge, PCI device 0)

A virtualized 440FX host bridge (vendor ID 0x8086, device ID 0x1237) is presented by the VMM, which allows it to present a virtualized PCI local bus. PCI configuration space accesses are intercepted by the virtual host bridge which allows other virtual and real PCI devices to be inserted into the virtual PCI local bus.

ISA Bridge (440FX ISA Bridge, PCI ID 8086/7000)

This virtual device is primarily used to route PCI interrupts to the interrupt controller, and to emulate the APM registers (mostly non-functional since power management is not emulated in the VM).

The VM provides four PCI IRQ links (PIRQA through PIRQD) which are routed via this device to inputs on the interrupt controller. The routing is initialized according to the VM configuration at startup, and dynamically controlled through the IRQ routing registers in this device.

EIDE Controller (PCI ID 8086/7010)

This device emulates the EIDE controller, providing EIDE functionality for the disk controller. The DMA functionality is provided by the VMM software.

ACPI Controller (PIIX4 ACPI Controller, PCI ID 8086/7113)

This device emulates the ACPI hardware controller for the chipset. Power management features are not emulated in the VM but all other ACPI hardware features are enabled.

B.3 Optional Virtual Machine Components**Ethernet controller**

An NE2000 virtual controller is provided which provides a shared-memory and non-interrupt driven Ethernet link to a Windows Ethernet driver.

Shared memory device

A shared memory device will be provided which allows for the development of a simple interface to a memory area shared between Windows and the guest operating system. A simple signaling mechanism is provided by the interface which generates interrupts from the virtual device. The Windows side of the interface is handled through an API provided in the product.

B.4 BIOS

The VM has a BIOS which provides the standard functionality of a PC BIOS, including all of the functions required to boot a guest operating system. The default action of the eVM is to execute the BIOS when first started, which then attempts to boot the guest operating system via the virtual IDE or floppy interface from a disk or CDROM image.

C. Guest OSes Currently Enabled:

The following OSes have been tested in basic configurations and the intention is to provide an adaptable configuration of the targeted guest OS with eVM. These need to support headless operation with a serial console mode.

- Microsoft Win CE 6.0
- QNX Neutrino
- WindRiver VxWorks
- SuSE Linux
- Siemens RMOS
- TenAsys iRMX

For an up-to-date listing of enabled guest OSes, check the following website: www.tenasys.com, and click on **Support->TenAsys Knowledge Base**.

D. VtProbe

VtProbe is one of the tools contained in the eVM installation. It is used during installation to determine if the hardware platform has the required virtualization features to support eVM. It can also be run standalone to manually check a hardware platform for eVM compatibility.

To run VtProbe, do a **Start->Run** and enter **vtprobe** or enter **vtprobe** in *Search programs and files*. If the platform does not support eVM, you will see that VT-x is not supported. In Figure D. 1, the hardware platform is found to support neither VT-x nor VT-d.

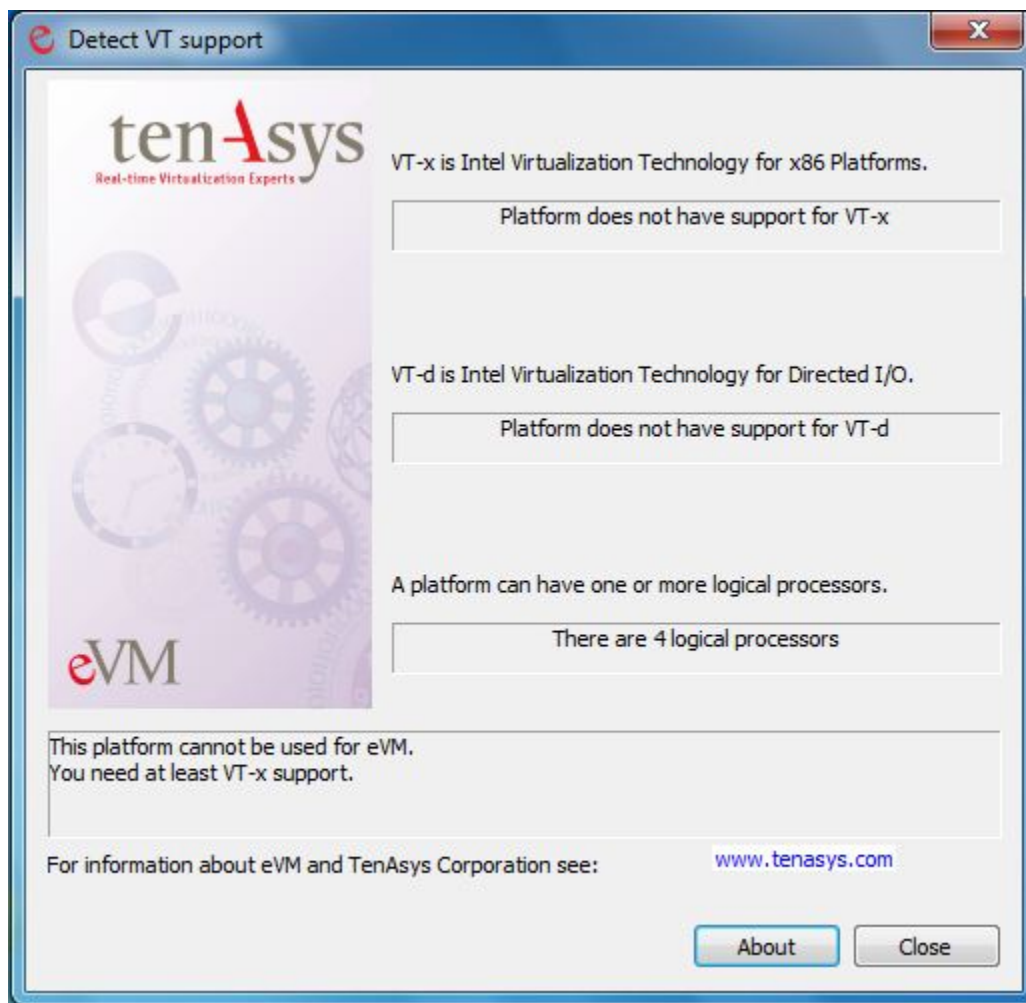


Figure D. 1 - VtProbe, Hardware Platform Doesn't Support eVM



Figure D. 2 - VtProbe: Hardware Supports Virtualization Technology

In Figure D. 2, the hardware platform was found to support both VT-x and VT-d.

E. Modifying device drivers for bus-master devices

Any device which generates bus cycles to transfer data is known as a bus master. Such a device is programmed with physical memory addresses at which it can read or write data. When a device driver running in the guest operating system programs such a device, either the platform must translate the physical addresses to direct the read/write operation to the correct physical memory address or the device driver must be modified to make this translation explicitly. This appendix describes how to make this modification.

E.1 Discovering the guest physical base address

The guest physical base address is the offset at which the guest is located on the host. This offset must be added to physical addresses programmed into any bus master device. This offset is discovered using an interface to the VMM via the VMCALL instruction. The VMCALL instruction causes an exit to the VMM which then processes the request based on the contents of the CPU registers. To find the guest physical address, execute the VMCALL instruction with EAX set to the value 0x80020001. If the call succeeds, then the carry flag will be clear, and the guest physical base address is returned in register EBX. The size of the memory allocated to the guest will be returned in register ECX. The VMCALL instruction is described in detail in Intel documentation.

E.2 Using the guest physical base offset value

Once the base address has been obtained, it can be used to offset any physical address generated by the device. Anywhere in a device driver where an address is provided which the device will use to generate a physical address has to be modified in this way.

A pseudocode implementation of a function to implement this might look like this (OsGetPhysicalAddress is assumed to be the operating system's regular call):

```
Paddr_t LibGetRealPhysicalAddress(PVOID *logical_address)
{
    Paddr_t real_address = OsGetPhysicalAddress(logical_address);
    UInt32 sys_offset ;
    UInt32 regs[4];

    regs[0] = GETMEMPARAMS;
    if (do_vmcall(regs) == 0)
        sys_offset = regs[1];
    else
        sys_offset = 0;

    return real_address + sys_offset;
}
```

F. Windows XP Installation of eVM for Windows

The steps for installing eVM for Windows on Windows XP are different that for Windows 7/Vista, due to differences in services provided and differences in the GUI for Windows XP versus Windows 7/Vista. The following provides a step-by-step guide of the installation screens with comments as necessary for Windows Xp, including eVM setup and License activation.

1. Run **evm100-install.exe** which begins the installation process. Windows XP will prompt if you want to run the installer, Figure F. 1. Click **Run**.



Figure F. 1 - Run evm100-install.exe Installer

2. Windows XP required a number of support upgrades, i.e. .NET Framework, Visual Studio 2008 Redistributable Package, and an upgrade to the Microsoft Installer. The eVM Installer prompts you that this will be done automatically for you and that the installation will require a reboot of the system for all changes to take effect.

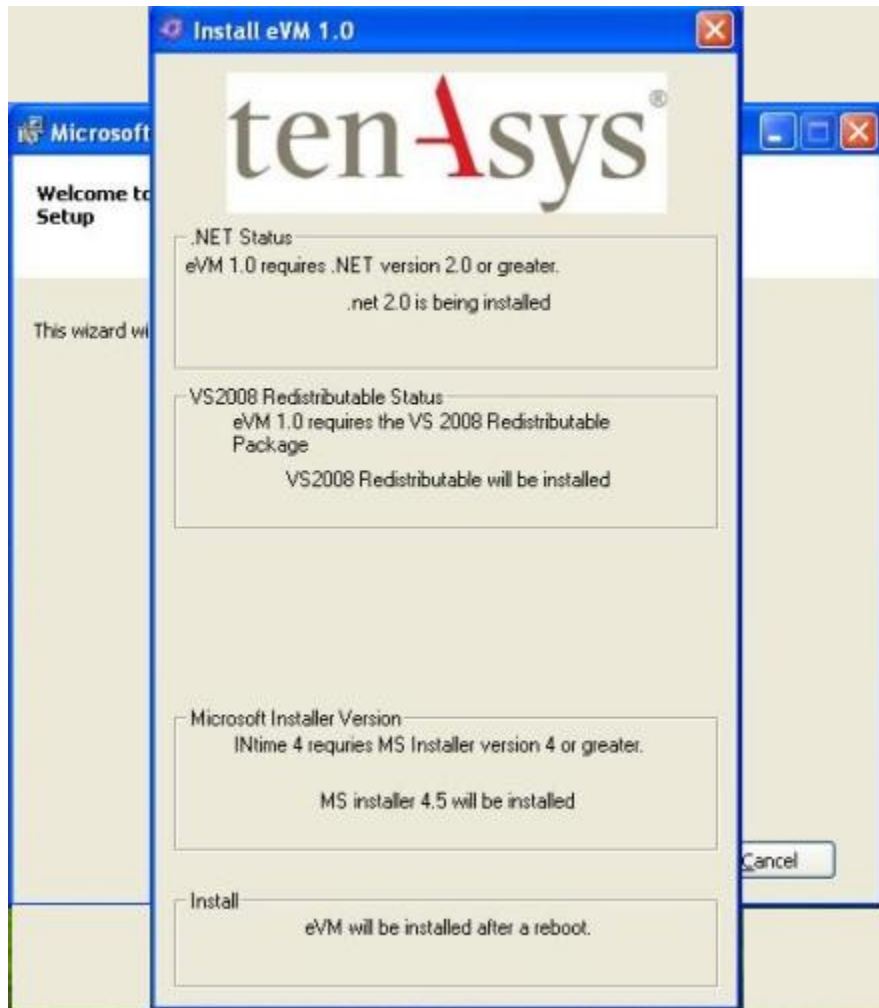


Figure F. 2 - eVM Installer System Upgrade Status Dialog

3. If the .NET Framework 2.0 Setup dialog is not the top dialog, click on it to give it the focus. Click **Next** to install the .NET Framework 2.0 setup, Figure F. 3.

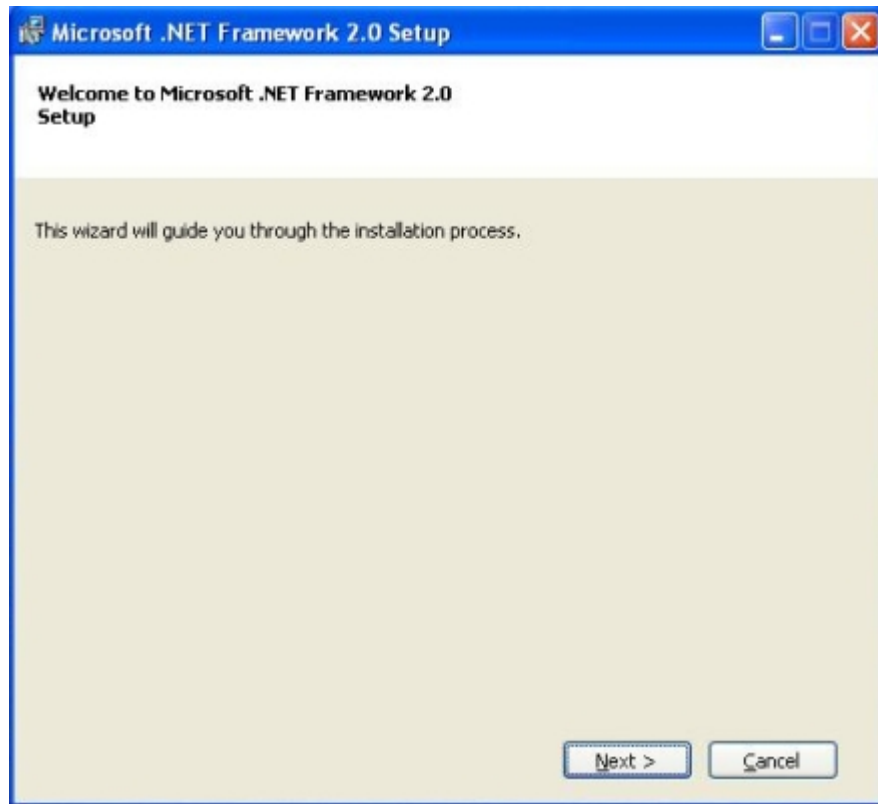


Figure F. 3 - .NET Framework 2.0 Setup Dialog Box

4. Read and accept the End-User License Agreement, Figure F. 4, and click Install.

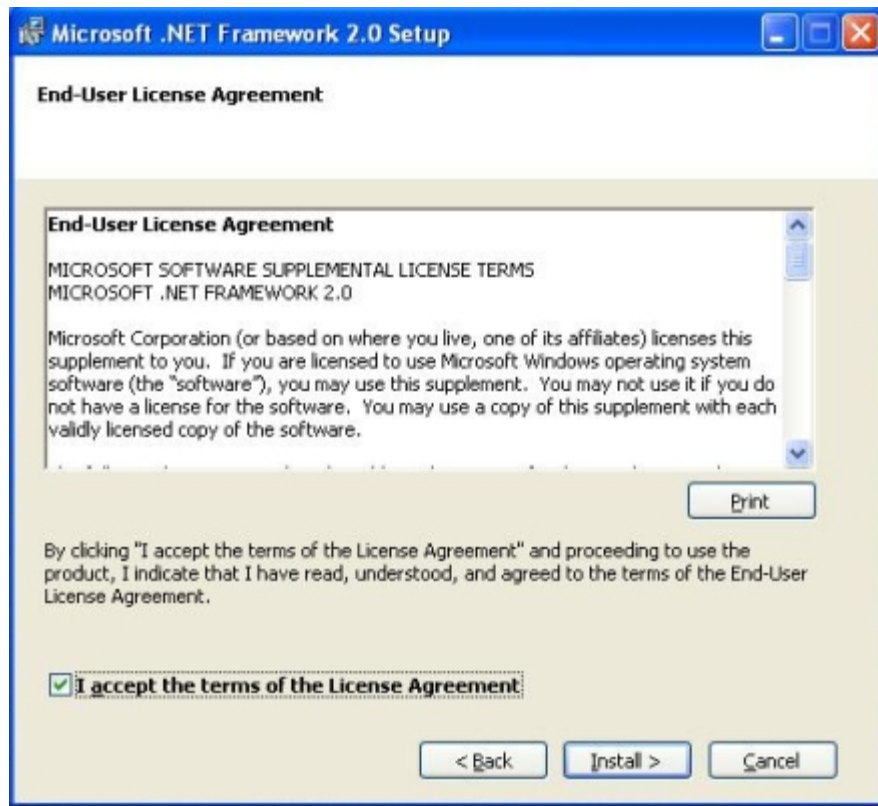
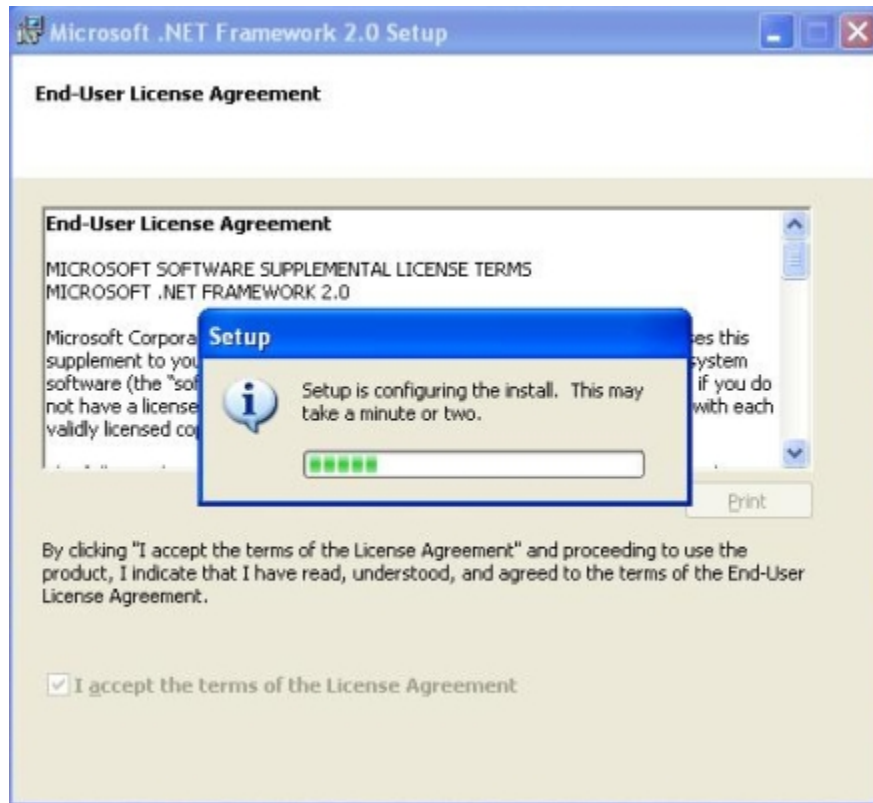


Figure F. 4 - .NET Framework 2.0 End-User License Agreement

5. A progress dialog, Figure F. 5, will be displayed as .NET Framework 2.0 is installed.



Followed by:

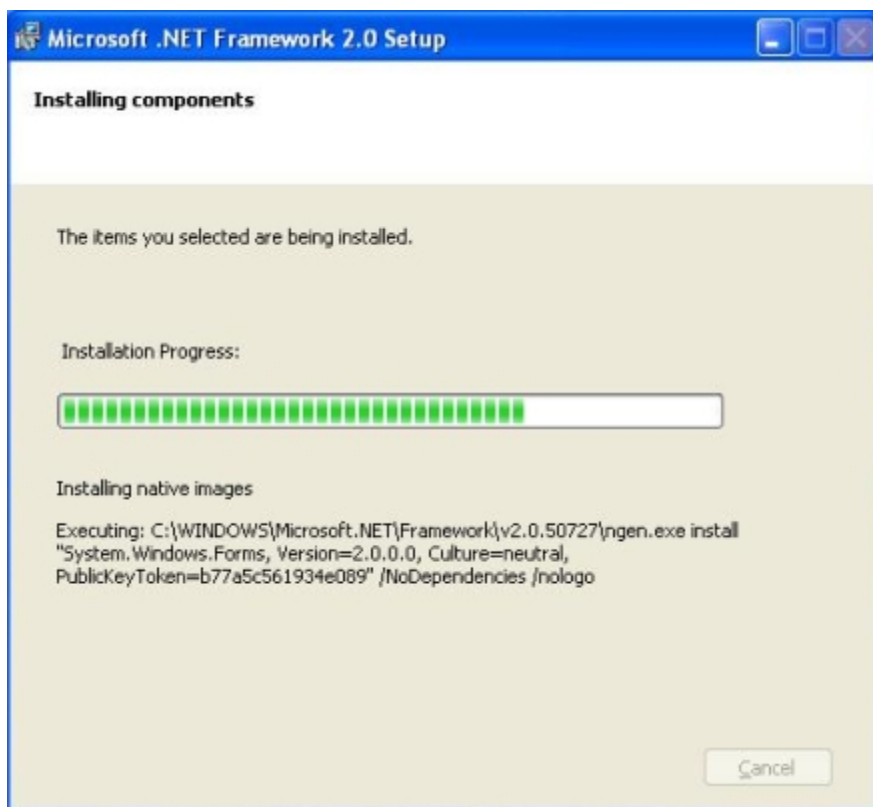


Figure F. 5 - .NET Framework 2.0 Installation Progress Dialog

6. When the installation completes successfully, Figure F. 6, click **Finish** to continue with the installation.

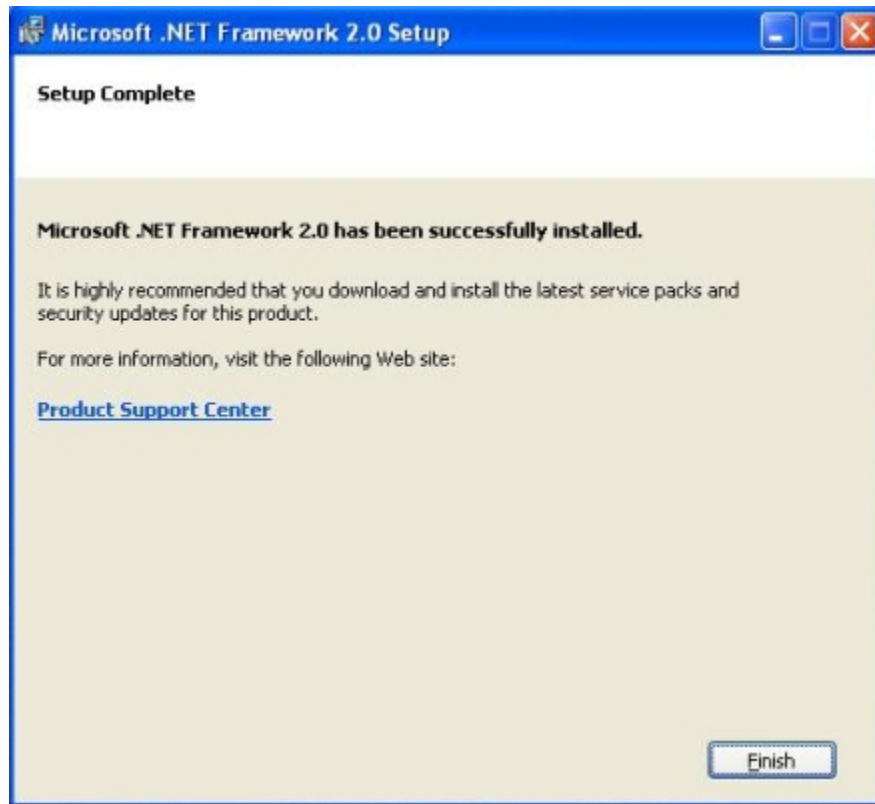


Figure F. 6 - .NET Framework 2.0 Setup Complete

7. Next, the installer will install the Microsoft Visual C++ 2008 Redistributable, Figure F. 7. Click **Next**.



Figure F. 7 - Installing Visual C++ 2008 Redistributable

8. Read and accept the Microsoft Visual C++ 2008 Redistributable license by checking the checkbox, Figure F. 8. Then click **Install**.

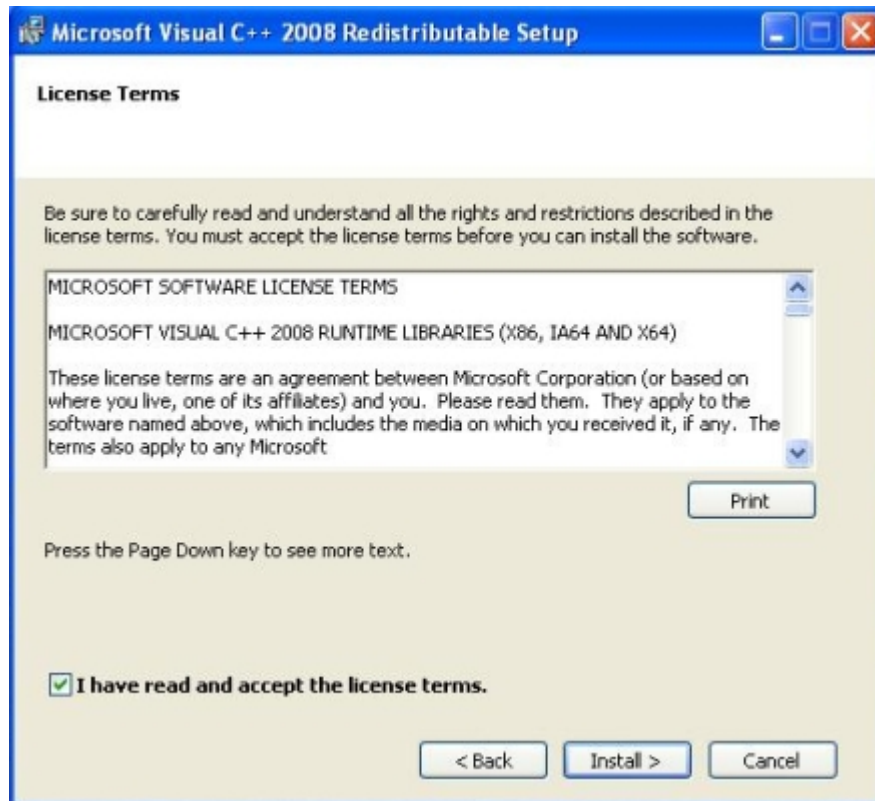


Figure F. 8 - Accept the Microsoft Visual C++ 2008 Redistributable License

9. A progress dialog will show the redistributable installation and setup. When it is complete, the following dialog in Figure F. 9 will be displayed. Click **Finish** to continue with the installation.



Figure F. 9 - Visual C++ 2008 Redistributable Setup Complete

10. Next, the installer needs to update the Microsoft Installer with a Hotfix for Windows XP, Figure F. 10. This will provides the support services for the eVM Installation Wizard. Click Next to install the Hotfix.



Figure F. 10 - Windows Installer Hotfix

11. Read and accept the license agreement for the Windows Installer Hotfix by clicking the **I Agree** radio button, Figure F. 11, and then click Next to continue with the installation.

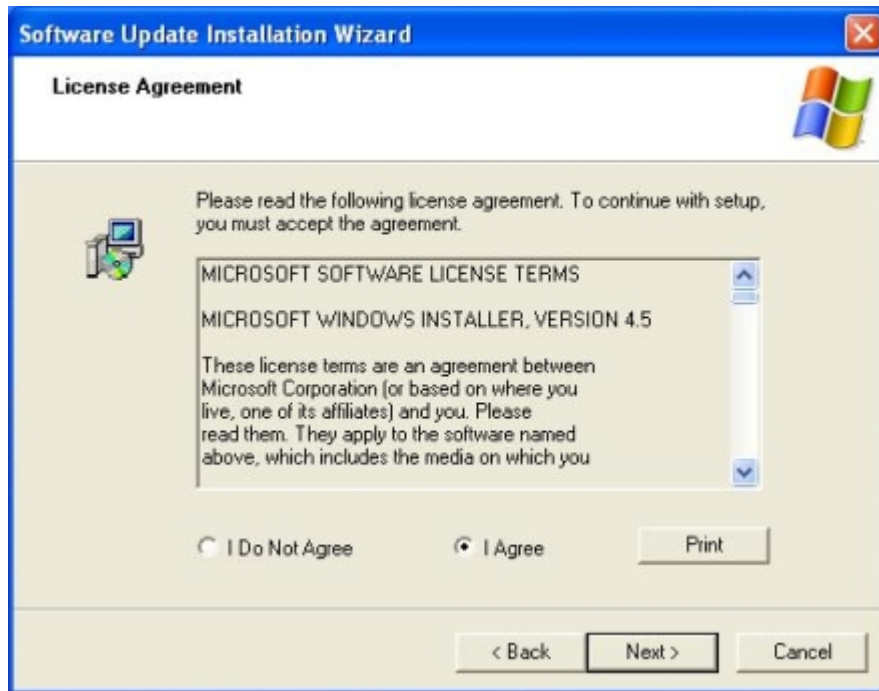


Figure F. 11 - Windows Installer License Agreement

12. A progress dialog will show the progress of the Windows Installer Hotfix installation. Then the installer will prompt to allow the Windows Installer Service to be stopped and restarted to affect the update . Click Continue to continue with the installation.

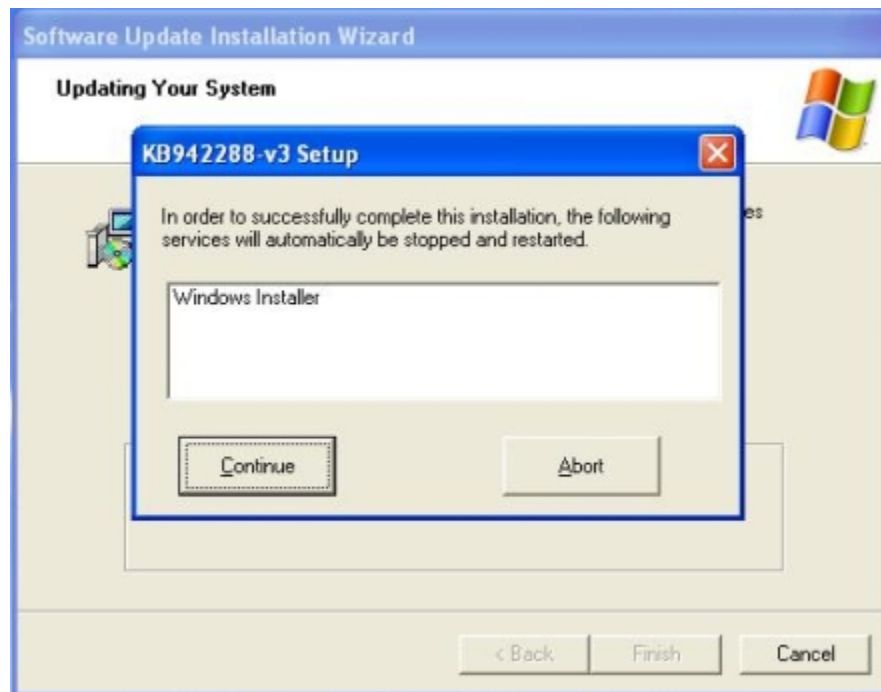


Figure F. 12 - Window Installer Service Restart Prompt

13. A service management dialog box will be displayed, Figure F. 13, as the update is managed .

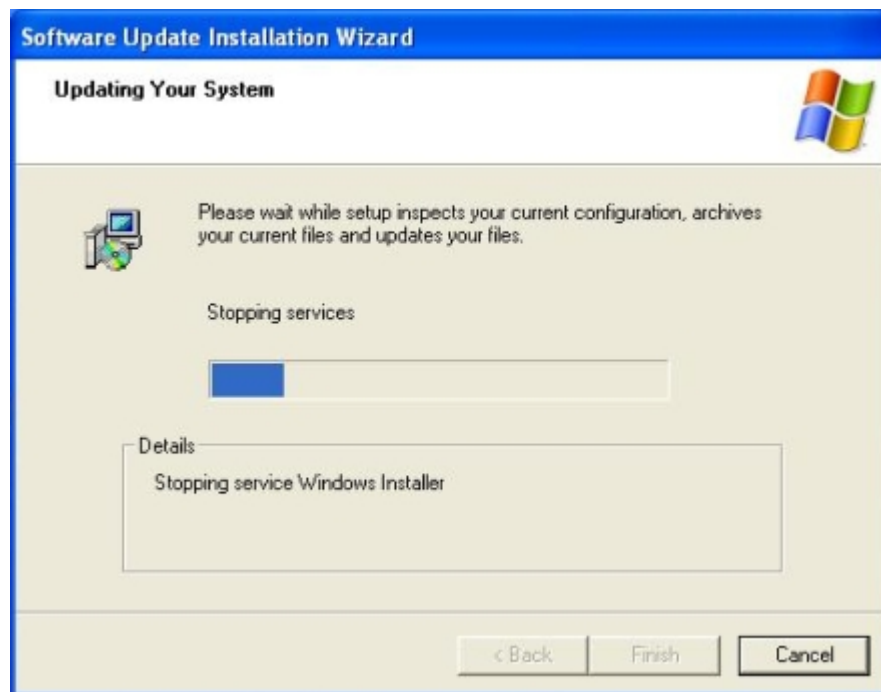


Figure F. 13 - Windows Installer Service Update Dialog

14. When the Windows Installer Hotfix has been successfully installed, the installer will prompt you telling you of the successful completion and that the system needs to be restarted for the changes to take effect, Figure F. 14. Leave the checkbox unchecked, and click **Finish**. The system will be rebooted and the eVM installation will automatically continue.



Figure F. 14 - Windows Install Hotfix Completed Successfully

15. After the system reboots, you will be prompted to continue the eVM installation, Figure F. 15. Click **Yes**.

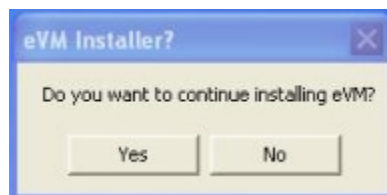


Figure F. 15 - Continue eVM for Windows Installation after Reboot

16. With all the Windows services and upgrades successfully installed, the eVM Installation Wizard is automatically launched, Figure F. 16. Click **Next**.



Figure F. 16 - eVM Installation Wizard Starting

17. Read and accept the TenAsys eVM Software License Agreement; and click **Next** to continue, Figure F. 17.



Figure F. 17 - TenAsys eVM Software License Agreement

18. Select the folder where you would like the installer to put the eVM installation files. The default location is "C:\Program Files\eVM\", Figure F. 18. Then click **Next** to continue.



Figure F. 18 - Select eVM Installation Folder

19. Select the type of installation that you would like. If you choose *Standard Installation*, Figure F. 19, eVM will be installed with virtual port support. If you choose *Complete Installation*, Figure F. 20, eVM with virtual port support, all the utilities, and a sample guest OS, iRMX will be installed. You can also choose a custom installation. After selecting the installation type, click **Next** to continue.

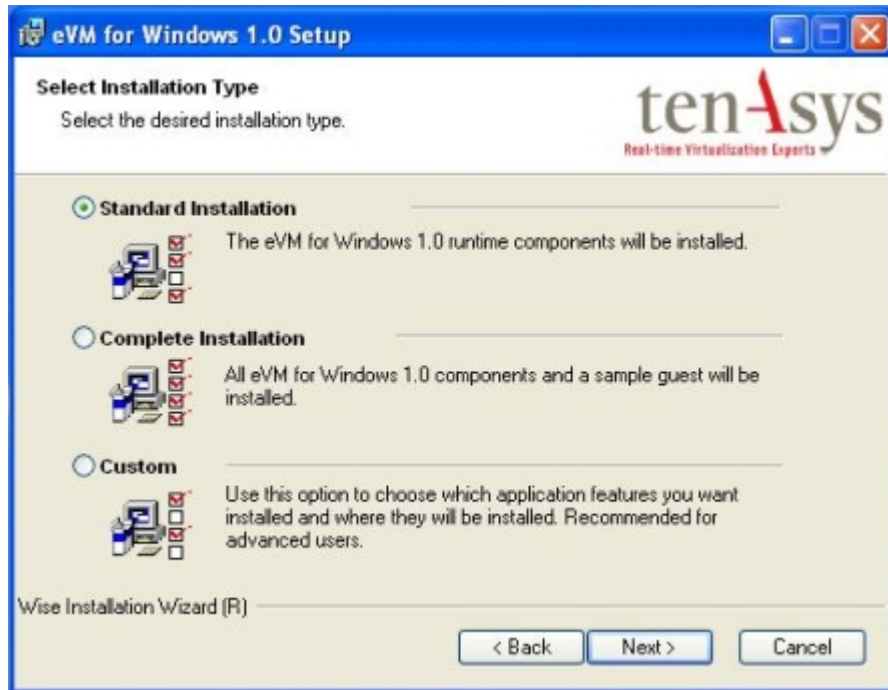


Figure F. 19 - Choosing Standard Installation Type



Figure F. 20 - Choosing Complete Installation Type

20. The eVM Installation Wizard will now prompt you to enter your name, organization, and your eVM activation code that you received when you purchased and downloaded eVM, Figure F. 21. Make sure you fill out all the fields properly and **be sure that you are connected to the Internet, as the activation code will be verified with TenAsys over the Internet**. Then Click **Next** to proceed with the installation.



Figure F. 21 - eVM Activation

21. When your activation code has been verified, the eVM Installation Wizard will then prompt you to confirm that you want to proceed with the installation, Figure F. 22. If you want to check or change anything at this point, you can click the **Back** button.

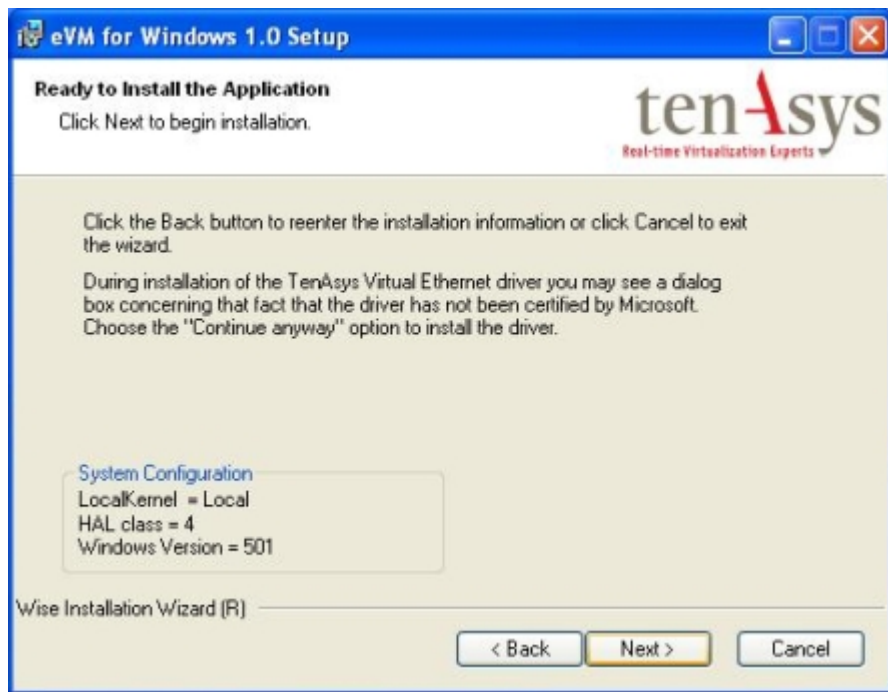


Figure F. 22 - eVM Installation Verification

To proceed with the installation, click **Next**.

22. You will then see a progress dialog, Figure F. 23, the will show you the status of the components and features that are being installed.



Figure F. 23 - eVM Installation Wizard Progress Dialog

23. The eVM Installation Wizard will next install the virtual Ethernet device. Windows XP will prompt and ask if you want it proceed with that device installation, Figure F. 24. Click **Continue Anyway** to install the device and continue with the installation.

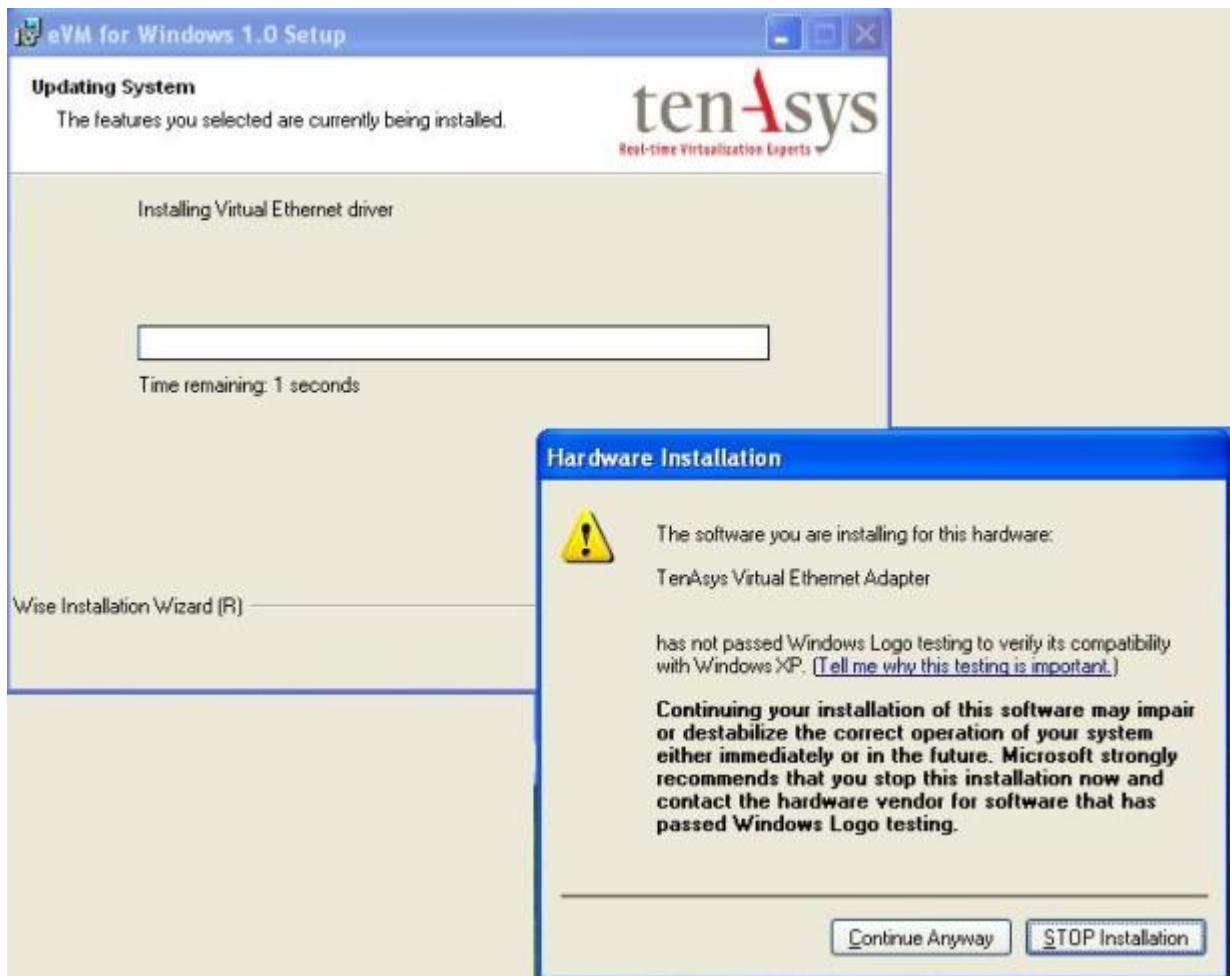


Figure F. 24 - Virtual Ethernet Driver Installation

24. The eVM Installation Wizard will next install the COM Emulator driver for virtual serial port devices, Figure F. 25. There are 4 virtual serial ports supported, so **you will be prompted to continue with the driver installation four times**. Click **Continue Anyway** for each of the hardware installation prompts to install all 4 virtual devices and continue with the eVM for Windows installation.

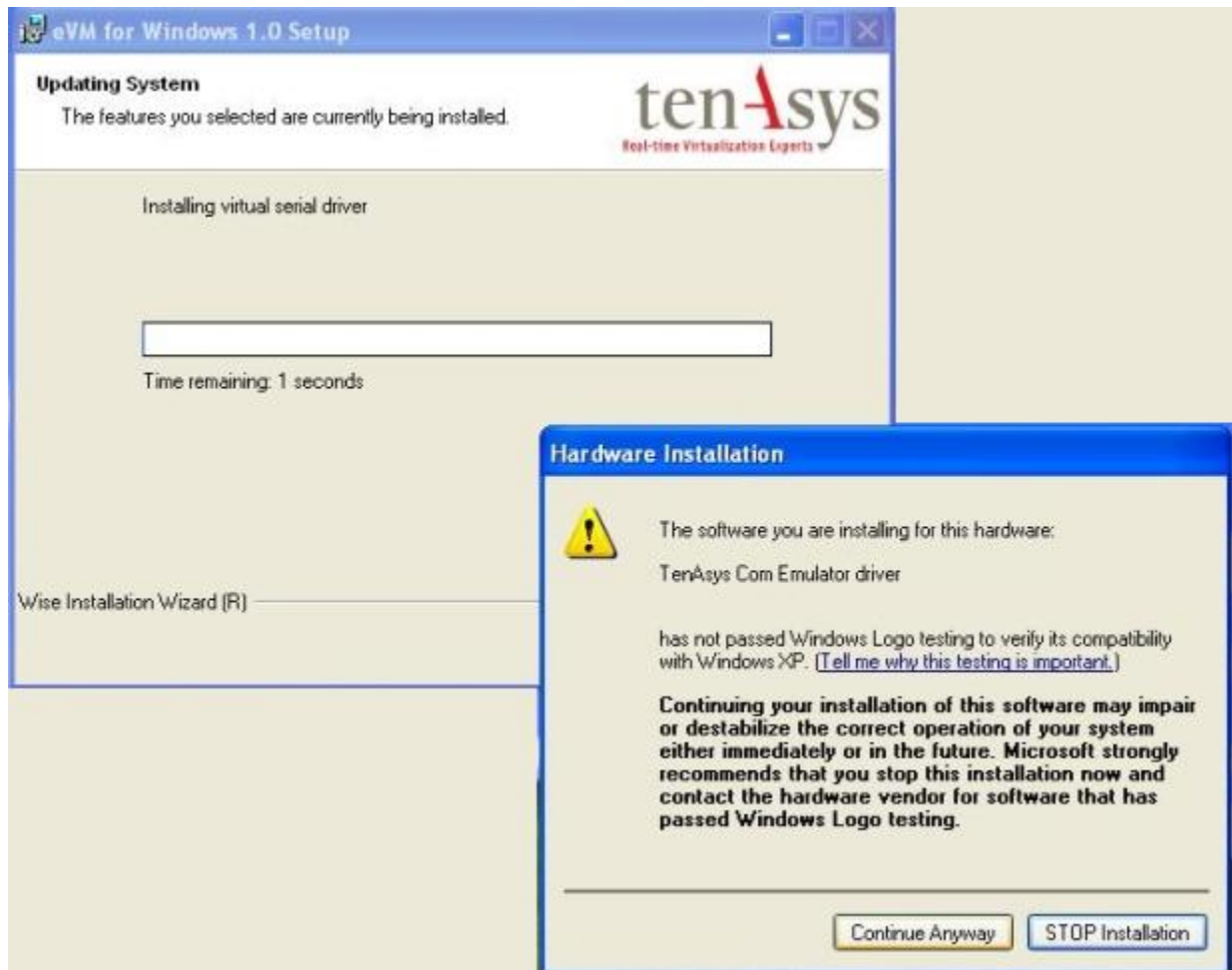


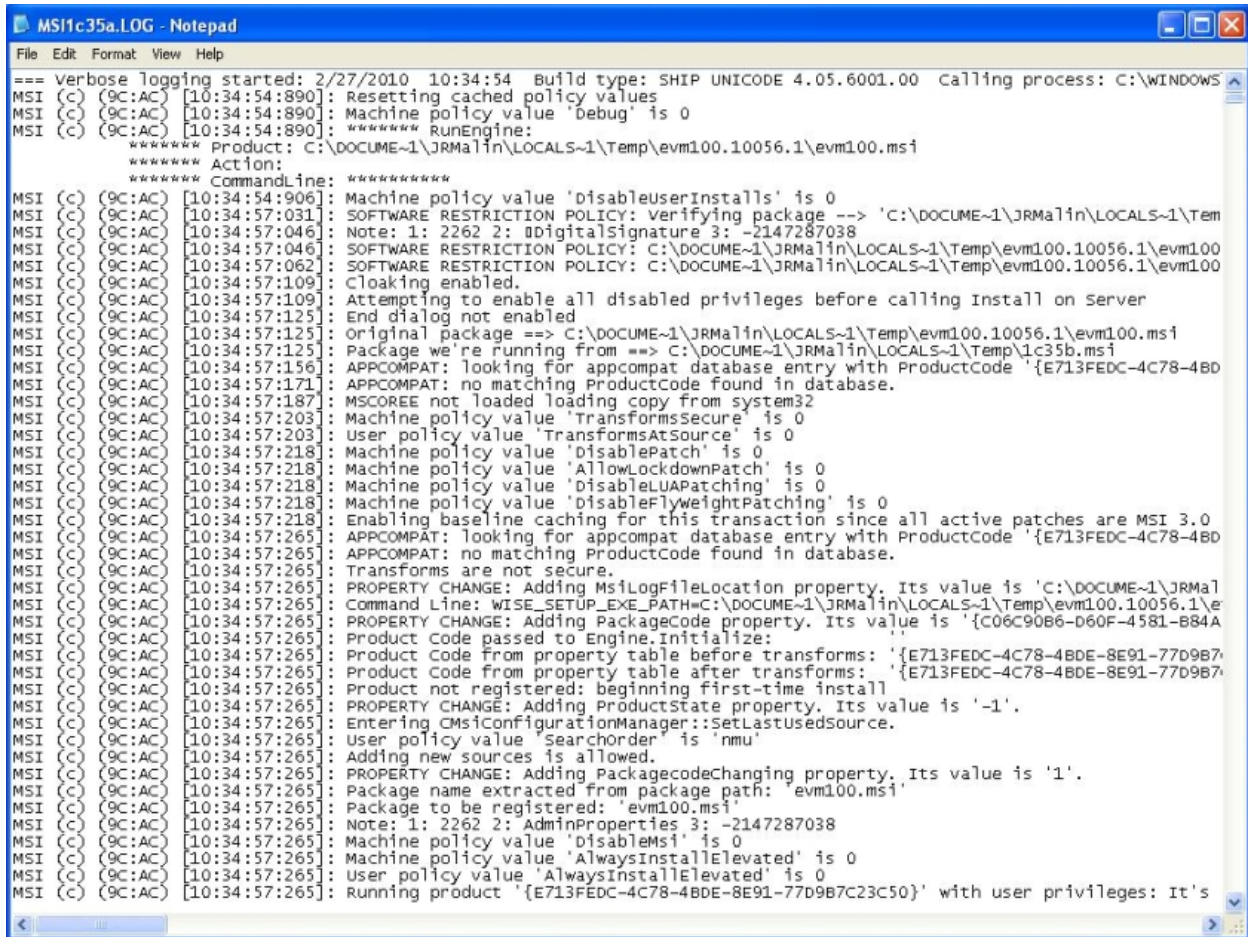
Figure F. 25 - Com Emulator Driver Installation

25. When all components and features of eVM that you selected have been installed, the Wizard will indicate that the installation was completed successfully, Figure F. 26. If you want to view the installation log to see what the Wizard did, check the **checkbox** to view the log. Whether you view the log or not, note the path where the log is saved. If you experience difficulties with the installation that you cannot resolve, TenAsys technical support may ask to see this log to help determine the problem. Now click **Finish**.



Figure F. 26 - eVM Installation Wizard Successful Installation

26. If you chose to view the installation log, you will see something similar to Figure F. 27. After viewing the log, you can close Notepad.



```

=== verbose logging started: 2/27/2010 10:34:54 Build type: SHIP UNICODE 4.05.6001.00 Calling process: C:\WINDOWS
MSI (C) (9C:AC) [10:34:54:890]: Resetting cached policy values
MSI (C) (9C:AC) [10:34:54:890]: Machine policy value 'debug' is 0
MSI (C) (9C:AC) [10:34:54:890]: ***** RunEngine:
***** Product: C:\DOCUME~1\JRMali\LOCALS~1\Temp\evm100.10056.1\evm100.msi
***** Action:
***** CommandLine: *****
MSI (C) (9C:AC) [10:34:54:906]: Machine policy value 'DisableUserInstallIs' is 0
MSI (C) (9C:AC) [10:34:57:031]: SOFTWARE RESTRICTION POLICY: Verifying package --> 'C:\DOCUME~1\JRMali\LOCALS~1\Temp\evm100.10056.1\evm100.msi'
MSI (C) (9C:AC) [10:34:57:046]: Note: 1: 2262 2: 0DigitalSignature 3: -2147287038
MSI (C) (9C:AC) [10:34:57:046]: SOFTWARE RESTRICTION POLICY: C:\DOCUME~1\JRMali\LOCALS~1\Temp\evm100.10056.1\evm100.msi
MSI (C) (9C:AC) [10:34:57:062]: SOFTWARE RESTRICTION POLICY: C:\DOCUME~1\JRMali\LOCALS~1\Temp\evm100.10056.1\evm100.msi
MSI (C) (9C:AC) [10:34:57:109]: Cloaking enabled.
MSI (C) (9C:AC) [10:34:57:109]: Attempting to enable all disabled privileges before calling Install on Server
MSI (C) (9C:AC) [10:34:57:125]: End dialog not enabled
MSI (C) (9C:AC) [10:34:57:125]: Original package ==> C:\DOCUME~1\JRMali\LOCALS~1\Temp\evm100.10056.1\evm100.msi
MSI (C) (9C:AC) [10:34:57:125]: Package we're running from ==> C:\DOCUME~1\JRMali\LOCALS~1\Temp\ic35b.msi
MSI (C) (9C:AC) [10:34:57:156]: APPCOMPAT: looking for appcompat database entry with ProductCode '{E713FEDC-4C78-4BD
MSI (C) (9C:AC) [10:34:57:171]: APPCOMPAT: no matching ProductCode found in database.
MSI (C) (9C:AC) [10:34:57:187]: MSCOREE not loaded loading copy from system32
MSI (C) (9C:AC) [10:34:57:203]: Machine policy value 'TransformsSecure' is 0
MSI (C) (9C:AC) [10:34:57:203]: User policy value 'TransformsAtSource' is 0
MSI (C) (9C:AC) [10:34:57:218]: Machine policy value 'DisablePatch' is 0
MSI (C) (9C:AC) [10:34:57:218]: Machine policy value 'AllowLockdownPatch' is 0
MSI (C) (9C:AC) [10:34:57:218]: Machine policy value 'DisableLUAPatching' is 0
MSI (C) (9C:AC) [10:34:57:218]: Machine policy value 'DisableFlyweightPatching' is 0
MSI (C) (9C:AC) [10:34:57:218]: Enabling baseline caching for this transaction since all active patches are MSI 3.0
MSI (C) (9C:AC) [10:34:57:265]: APPCOMPAT: looking for appcompat database entry with ProductCode '{E713FEDC-4C78-4BD
MSI (C) (9C:AC) [10:34:57:265]: APPCOMPAT: no matching ProductCode found in database.
MSI (C) (9C:AC) [10:34:57:265]: Transforms are not secure.
MSI (C) (9C:AC) [10:34:57:265]: PROPERTY CHANGE: Adding MSILogFileLocation property. Its value is 'C:\DOCUME~1\JRMali
MSI (C) (9C:AC) [10:34:57:265]: Command Line: WISE_SETUP_EXE_PATH=C:\DOCUME~1\JRMali\LOCALS~1\Temp\evm100.10056.1\ev
MSI (C) (9C:AC) [10:34:57:265]: PROPERTY CHANGE: Adding PackageCode property. Its value is '{C06C90B6-D60F-4581-B84A
MSI (C) (9C:AC) [10:34:57:265]: Product Code passed to Engine.Initialize:
MSI (C) (9C:AC) [10:34:57:265]: Product Code from property table before transforms: '{E713FEDC-4C78-4BDE-8E91-77D9B7
MSI (C) (9C:AC) [10:34:57:265]: Product Code from property table after transforms: '{E713FEDC-4C78-4BDE-8E91-77D9B7
MSI (C) (9C:AC) [10:34:57:265]: Product not registered: beginning first-time install
MSI (C) (9C:AC) [10:34:57:265]: PROPERTY CHANGE: Adding ProductState property. Its value is '-1'.
MSI (C) (9C:AC) [10:34:57:265]: Entering CMSiConfigurationManager::SetLastUsedSource.
MSI (C) (9C:AC) [10:34:57:265]: User policy value 'SearchOrder' is 'nmu'
MSI (C) (9C:AC) [10:34:57:265]: Adding new sources is allowed.
MSI (C) (9C:AC) [10:34:57:265]: PROPERTY CHANGE: Adding PackageCodeChanging property. Its value is '1'.
MSI (C) (9C:AC) [10:34:57:265]: Package name extracted from package path: 'evm100.msi'
MSI (C) (9C:AC) [10:34:57:265]: Package to be registered: 'evm100.msi'
MSI (C) (9C:AC) [10:34:57:265]: Note: 1: 2262 2: AdminProperties 3: -2147287038
MSI (C) (9C:AC) [10:34:57:265]: Machine policy value 'DisableMSI' is 0
MSI (C) (9C:AC) [10:34:57:265]: Machine policy value 'AlwaysInstallElevated' is 0
MSI (C) (9C:AC) [10:34:57:265]: User policy value 'AlwaysInstallElevated' is 0
MSI (C) (9C:AC) [10:34:57:265]: Running product '{E713FEDC-4C78-4BDE-8E91-77D9B7C23C50}' with user privileges: It's
  
```

Figure F. 27 - eVM Installation Wizard Installation Log

27. After you click **Finish** in the dialog shown in Figure F. 26, the Wizard will prompt you that you now need to restart your system, Figure F. 28. A restart is required, so eVM can take the memory that it needs for eVM and the guest OS from Windows. In general, resizing Windows available memory requires a restart of the Windows OS. Click **Yes**, and wait for the system to reboot.

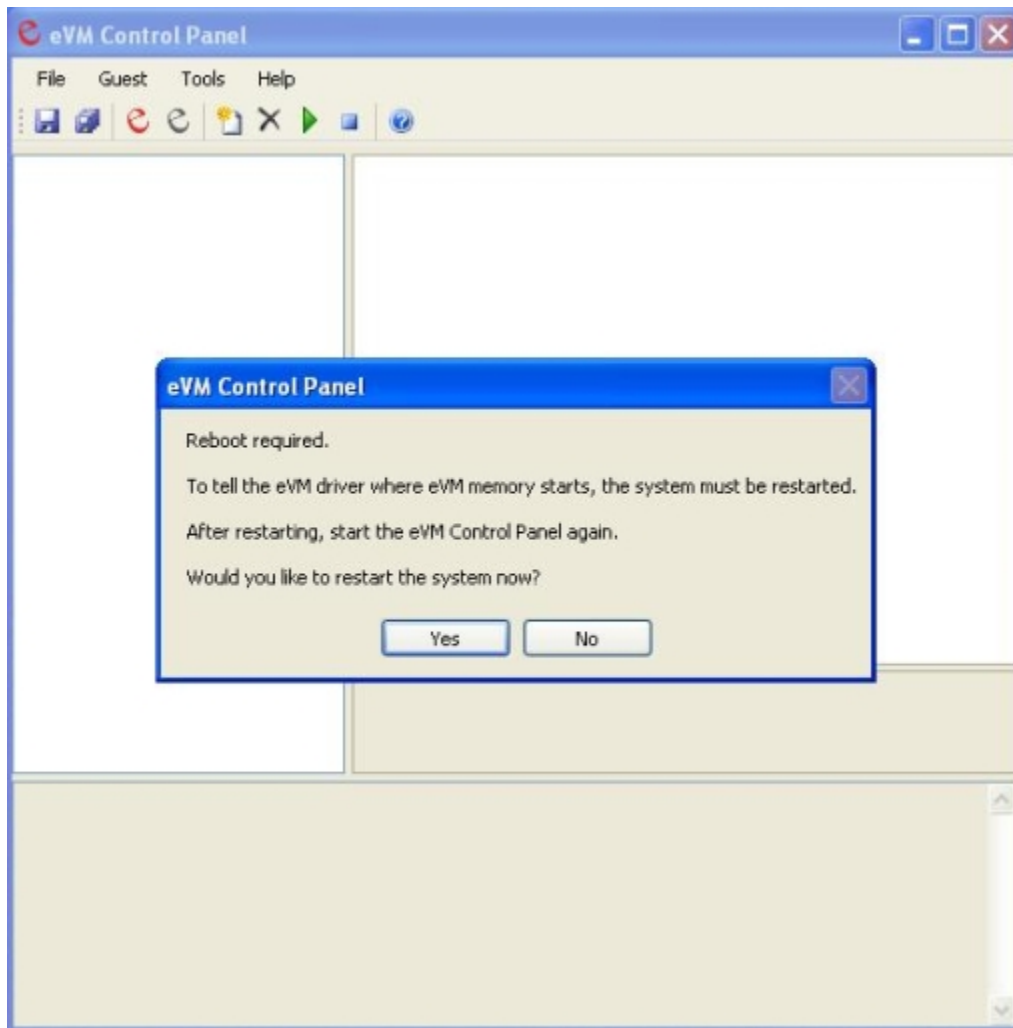


Figure F. 28 - eVM Control Panel Restart Prompt

28. After the system has rebooted, wait for the eVM services to start, which will be indicated by the 'e' icon showing up in the system tray, Figure F. 29.



Figure F. 29 - eVM System Tray Icon

29. Click on the **eVM tray icon** and select **eVM Control Panel** to launch the eVM Control Panel, Figure F. 30.



Figure F. 30 - Launching the eVM Control Panel from the system tray

30. When the eVM Control Panel launches, if you chose the *Standard Installation*, you will see the system configuration and no eVM guests installed. If you chose the *Complete Installation*, not only has eVM been installed with the 2 virtual serial ports, Figure F. 31, but also has the sample guest RTOS, iRMX, complete with a sample configuration been installed, Figure F. 32.

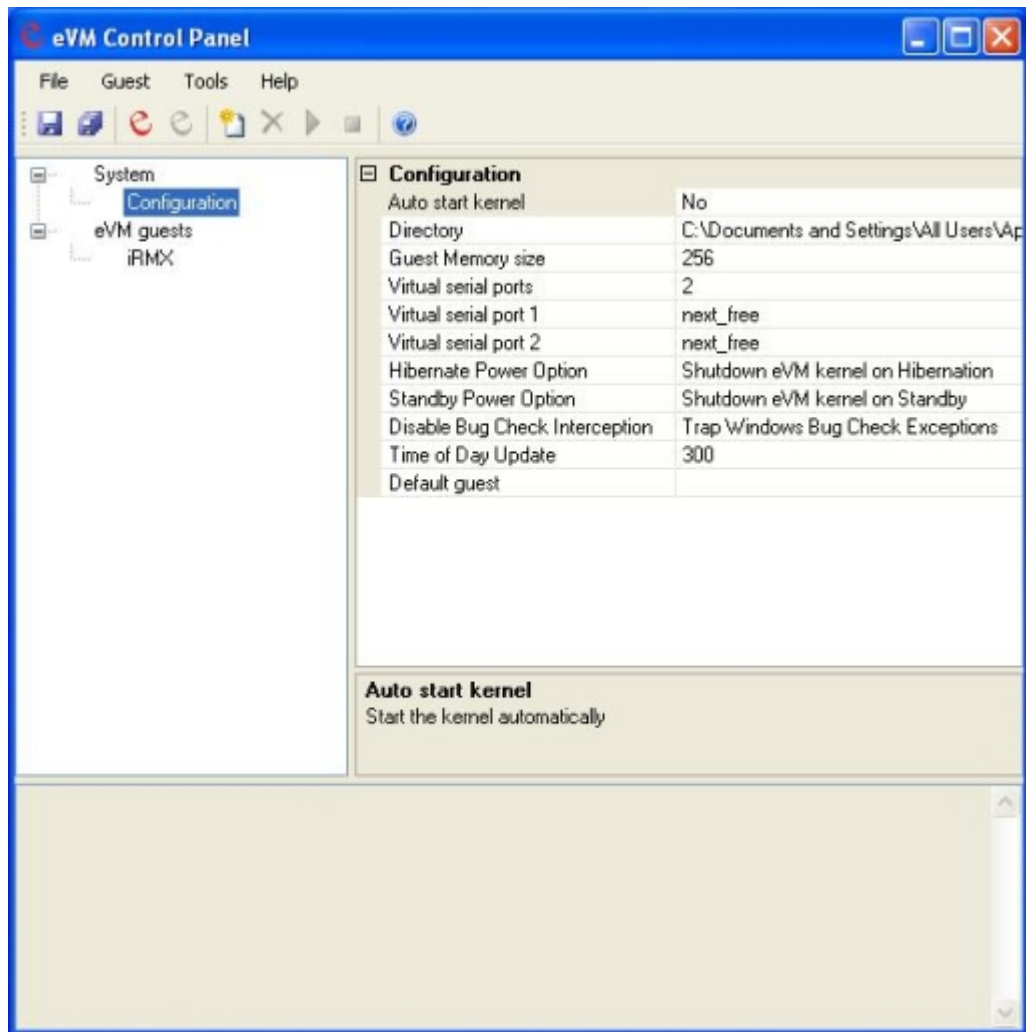


Figure F. 31 - Complete eVM Installation Showing System Configuration

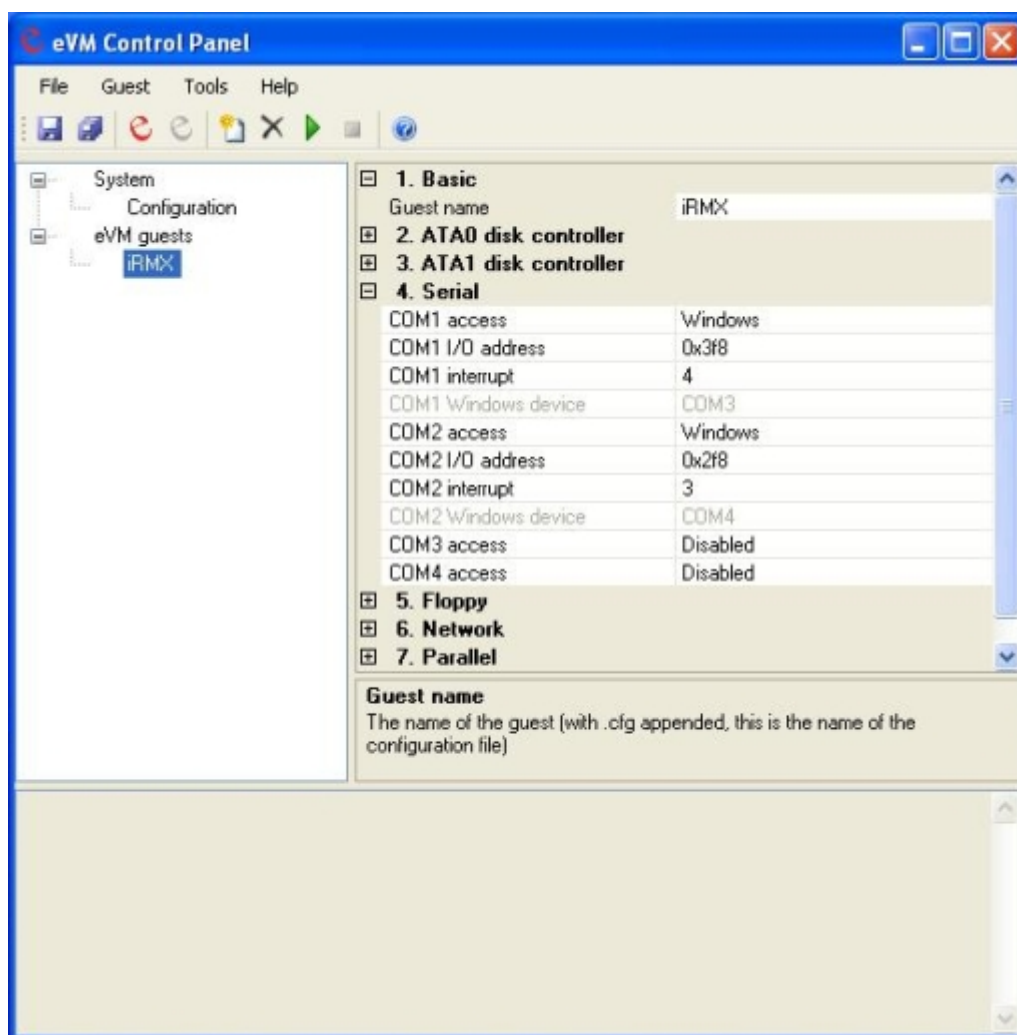


Figure F. 32 - Complete eVM Installation Showing iRMX Guest Configuration

31. Finally, let's look at the devices that are now in the Windows host system. Open Windows Device Manager by clicking Windows **start**, right-clicking **My Computer**, and clicking **Properties**. Then in the *System Properties* dialog, click the **Hardware** tab at the top and click the **Device Manager** button. If you expand the **Network adapters** node and **Ports (COM & LPT)** node, you can see that the TenAsys Virtual Ethernet Adapter has been installed, as well as, the two virtual serial ports, Figure F. 33. The property of each virtual serial port was to use the next available COM device, so you can see that in this system configuration, it resulted in the virtual serial ports becoming COM3 and COM4.

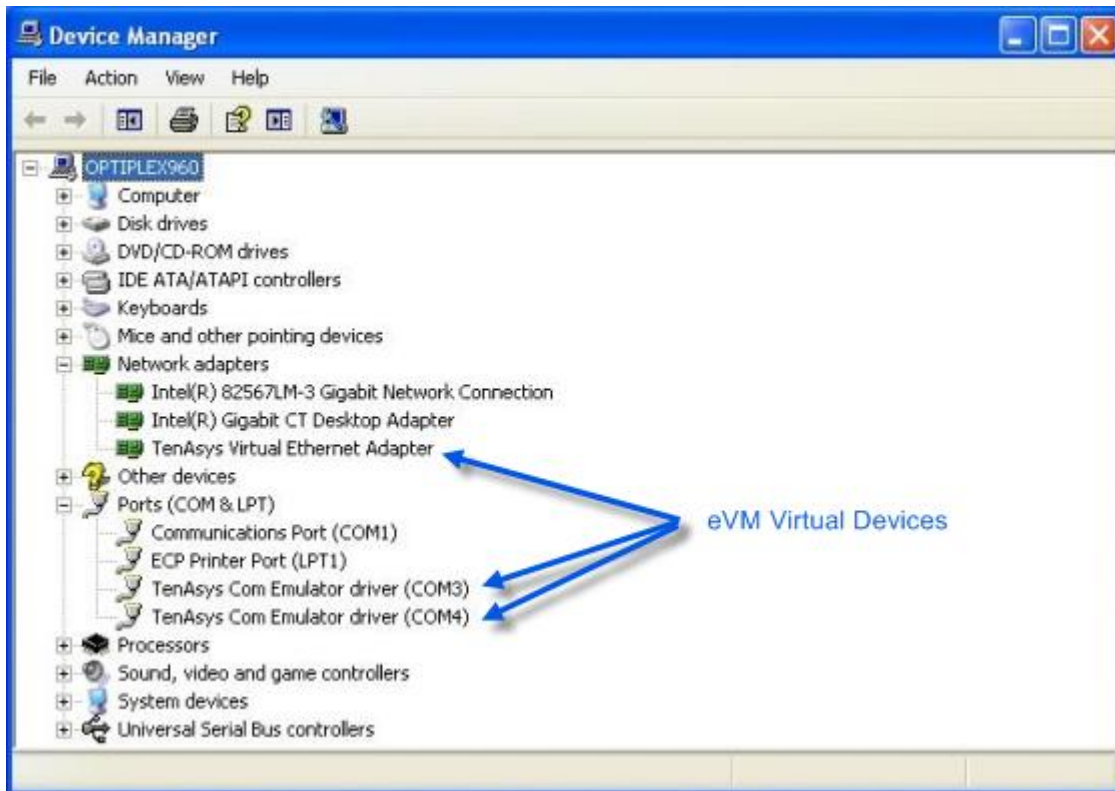


Figure F. 33 - Windows Device Manager Showing eVM Virtual Devices

G. Windows XP Configuration of the Virtual Ethernet Device

There are two parts to setting up eVM to manage the Virtual Ethernet connection, the virtual Ethernet device on the Windows side, and the virtual Ethernet device on the guest OS side. This appendix only describes the configuration steps for the Windows side when running on Windows XP.

1. Open the **Windows Control Panel** and double-click on **Network Connections**, Figure G. 1.

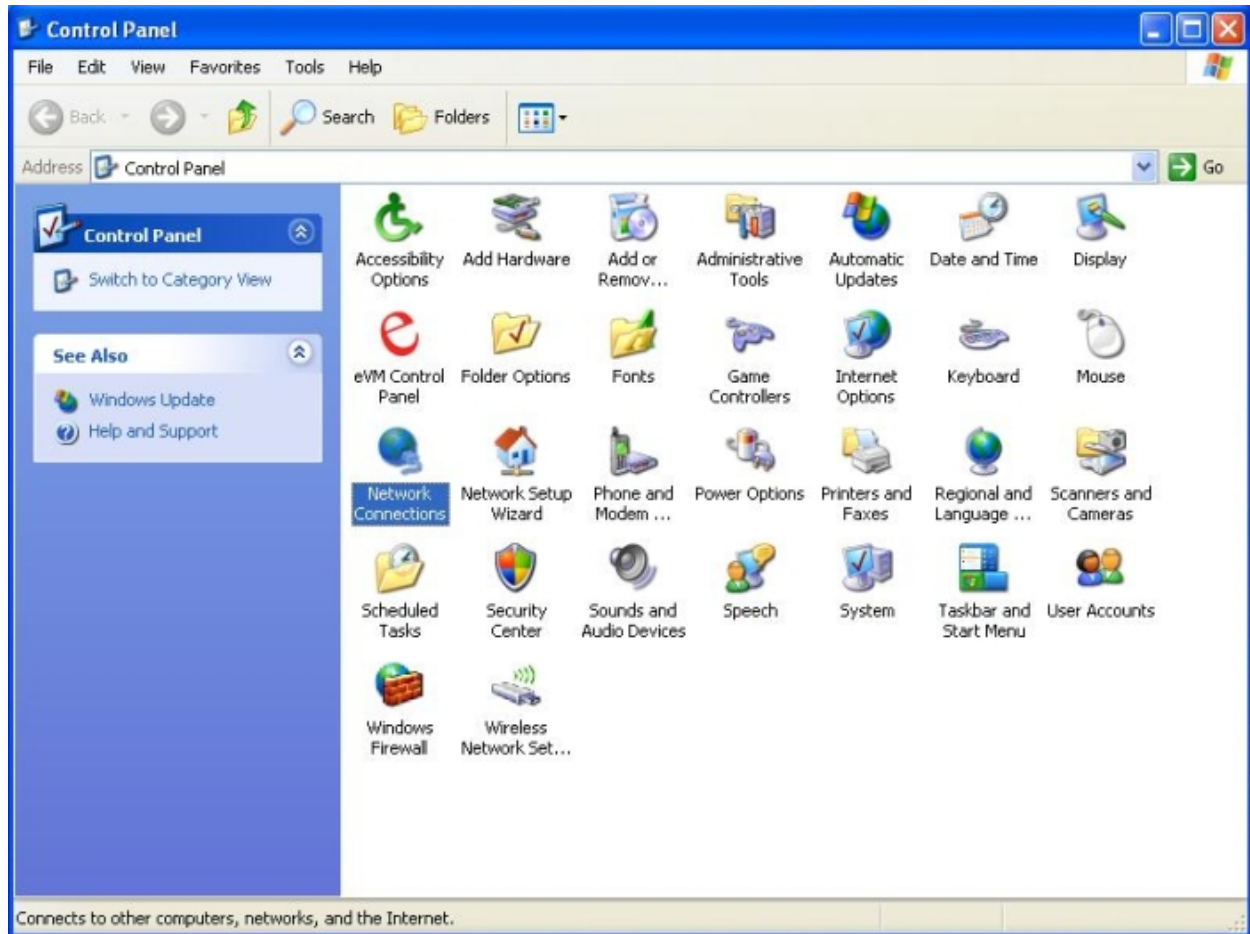


Figure G. 1 - Windows Control Panel, Select Network Connections

2. The *Network Connection* dialog will open; and if you browse the connections displayed, you will see the *TenAsys Virtual Ethernet Adapter* (in this example it is *Local Area Connection 3*), Figure G. 2. Double-click on that local area connection.

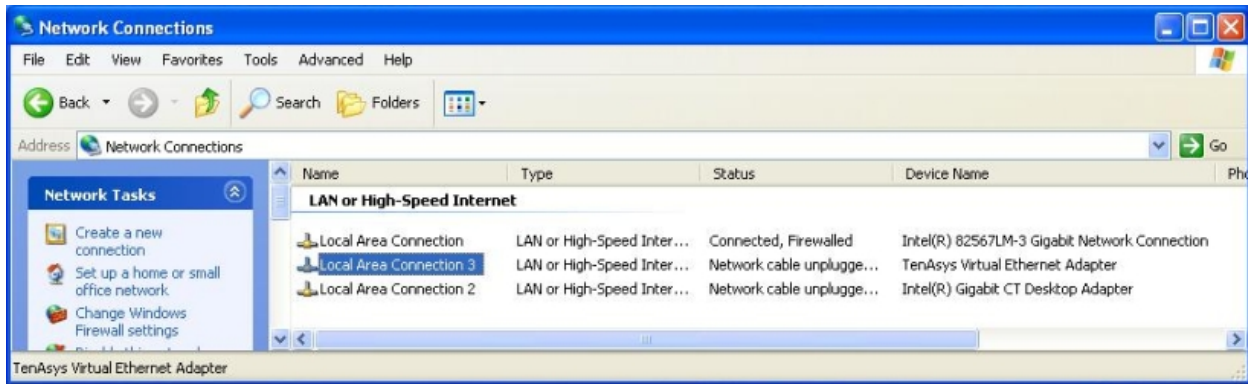


Figure G. 2 - Network Connections

3. This will open the *Local Area Connection Properties*. Click on **Internet Protocol (TCP/IP)**, and then click the **Properties** button, Figure G. 3, so that we can set up the TCP/IP properties of the virtual Ethernet device.

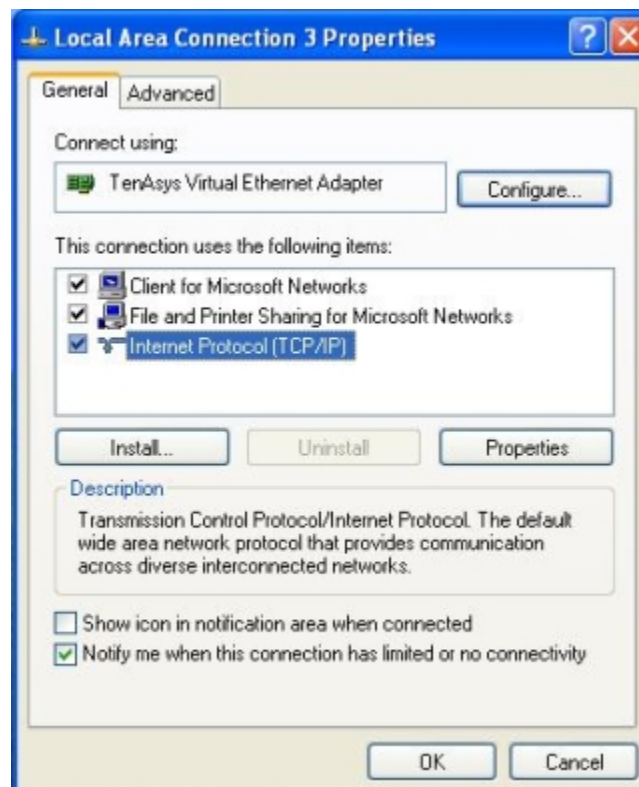


Figure G. 3 - TenAsys Virtual Ethernet Adapter Local Area Connection Properties

4. You will now see the *Internet Protocol (TCP/IP) Properties*, Figure G. 4. Select the **Use the following IP address:** radio button, and enter **10.0.0.1** in the IP address. Keep the default Subnet mask: of **255.0.0.0**, . Then click OK, to apply the new settings.



Figure G. 4 - Setting the Static IP Address for the TenAsys Virtual Ethernet Adapter

5. Close the *Local Area Connection Properties* window and the *Network Connections* window, and the Windows side of the virtual Ethernet connection configuration is complete.